# MeMo - Eine offene Integrationsplattform zur modellbasierten Entwicklung von Fahrzeugsystemen

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung der Würde

eines Doktor-Ingenieurs (Dr.-Ing.)

genehmigte Dissertation


| | |
|---|---|
| Von: | Dipl.-Ing. Daniel Brechter |
| Aus (Geburtsort): | Wangen im Allgäu |



| | |
|---|---|
| Eingereicht am: | 30. Juni 2008 |
| Mündliche Prüfung am: | 18. November 2008 |
| Referenten: | Prof. Dr.-Ing. Walter Schumacher<br>Prof. Dr.-Ing. Rolf Ernst |

2008

Die Meinungen, Ergebnisse und Schlüsse in dieser Dissertation sind nicht notwendigerweise die der Volkswagen AG.

# Vorwort

Die vorliegende Dissertation ist während meiner Tätigkeit in der Abteilung für Mechatronik und Methodenentwicklung bei der Volkswagen AG in einer Kooperation mit dem Institut für Regelungstechnik an der Technischen Universität Carolo-Wilhelmina zu Braunschweig entstanden.

Mein besonderer Dank gilt Herrn Prof. Dr.-Ing. Walter Schumacher, dem Leiter des Instituts für Regelungstechnik, der meine wissenschaftliche Arbeit förderte und durch zahlreiche fruchtbare Diskussionen anregte.

Für die Übernahme der Mitberichterstattung danke ich Herrn Prof. Dr.-Ing. Rolf Ernst. Herrn Prof. em. Dr.-Ing. Dr. h.c. Werner Leonhard danke ich für die Übernahme des Prüfungsvorsitzes.

Meinen Kollegen bei der Volkswagen AG sowie den Mitarbeitern bei der David GmbH danke ich für die Unterstützung durch wertvolle Anregungen und fachliche Diskussionen. Ein besonderer Dank gilt Frau Prof. Dr.-Ing. Xiaobo Liu-Henke für die fachliche und methodische Betreuung, die wesentlich zum Gelingen der Arbeit beigetragen hat. Ebenso danke ich Herrn Dipl.-Math. Thomas Salmikeit für das Korrekturlesen und die stets motivierenden Worte.

Ganz herzlich danken möchte ich auch meinen Eltern und nicht zuletzt meiner Frau Tanja, die mich während der Promotion geduldig motiviert und liebevoll unterstützt haben.

Wolfsburg, November 2008 Daniel Brechter

# Inhaltsverzeichnis

# 1 Einleitung

Im Produktlebenszyklus des Kraftfahrzeugs, in dem heute etwa 50% der Garantie- und Kulanzkosten mit Elektronik und Elektrik in Zusammenhang gebracht werden können, verspricht die modellbasierte Entwicklung und Absicherung von Steuergerätealgorithmen noch ein erhebliches Potential [Hol06][1], [Erl07].

In modernen Fahrzeugen werden heute 60 bis 100 Steuergeräte und mehr als 100 Megabyte[2] Software verbaut. Bis zu 10 Millionen Codezeilen müssen konfliktfrei interagieren. Dabei wird Hard- und Software von unterschiedlichen Herstellern mit unterschiedlichen Standards – zum Teil auch proprietäre Systeme – über mehrere Busse miteinander verbunden [Bay05]. Beispielsweise sind in einem aktuellen Oberklassefahrzeug ungefähr 1500 Softwarefunktionen implementiert, wobei knapp 500 Funktionen mehr als ein Steuergerät benötigen. Etwa 50 besonders komplexe Funktionen erfordern mehr als neun Steuergeräte [Pla06].

Dies macht deutlich, dass bereits heute im Automobilbau auf die Integration von elektronischen Systemen und eine breite virtuelle Absicherung nicht verzichtet werden kann. Vielmehr stellt diese eine der Schlüsseldisziplinen für den zukünftigen Automobilbau dar, da laut aktuellen Schätzungen zukünftig 90% aller Innovationen im Automobil in ihrer wesentlichen Funktionalität durch Elektronik ergänzt sein werden. Für die Realisierung wird dabei aus Kostengründen überwiegend auf Standard-Komponenten zurückgegriffen, während die eigentliche Funktionalität in der Software implementiert wird [Hof00], [Schi04], [Bay05].

Die Steuergerätefunktionen in Verbindung mit den im Fahrzeug vorhandenen Komponenten wie Aktorik, Sensorik und Mechanik sind zudem ein fester Bestandteil mechatronischer Systeme[3] im Kraftfahrzeug, die in annähernd allen Entwicklungsbereichen des Fahrzeuges eingesetzt werden. Neben dem Einsatz im Komfortbereich werden auch sicherheitsrelevante Bereiche im Fahrzeug unterstützt. Im Bereich der Fahrdynamik sind dies z. B. Funktionen wie ESP[4], EPS[5] und ACC[6], die hohen Sicherheitsanforderungen genügen müssen.

Anders als bei Funktionen, die als Steuerungen ausgelegt werden, ist bei Regelungen ein direkter Bezug zur zu regelnden Strecke bereits zu Beginn der Entwicklung unerlässlich. Bei mechatronischen Fahrzeugsystemen wird das Zusammenführen von Regelung und Strecke aufgrund des heterogenen Systemcharakters zusätzlich erschwert. Grund hierfür ist die dezentrale Entwicklung der Teilelemente in unterschiedlichen Organisationseinheiten

[1] Die auf die Abkürzungen bezogene Literatur befindet sich im Literaturverzeichnis ab Seite 111.

[2] Maßeinheit für die Datenmenge.

[3] Der Begriff „Mechatronik“ wurde ursprünglich 1969 in Japan durch die Zusammensetzung der Begriffe Mechanik und Elektronik gebildet [Kit86] [Bus93]. Mechatronik stellt heute ein interdisziplinäres Gebiet dar, bei dem typischerweise die Teilsysteme Mechanik, Elektronik und Informatik zusammenwirken [Ise99].

[4] ESP: Elektronisches Stabilitätsprogramm

[5] EPS: Elektromechanische Servolenkung

[6] ACC: Fahrerassistenzsystem zur Abstandsregelung vorausfahrender Fahrzeuge

der OEM[7] oder beim Lieferanten. In der Literatur wird bei mechatronischen Fahrzeugsystemen von einem notwendigen integrativen Entwicklungscharakter gesprochen, durch den eine systematische Integration der heterogenen Systeme zu einem frühen Zeitpunkt und über den kompletten Entwicklungszyklus zu gewährleisten ist [VDI06], [Kal97].
Die vorliegende Arbeit soll einen Beitrag dazu leisten, den integrativen Entwicklungscharakter mechatronischer Fahrzeugsysteme besser zu unterstützen.
Dies bedeutet zum einen, dass die Entwicklung mechatronischer Fahrzeugsysteme bzgl. Hierarchisierung und Modularisierung, Kommunikation und Kooperation sowie dem parallelen Zusammenarbeiten mehrerer Fachleute aus unterschiedlichen Domänen gewährleistet sein muss. Zum anderen steht die Funktionsabsicherung von Steuergeräten als Bestandteil mechatronischer Fahrzeugsysteme im Fokus und muss bestmöglich in einer methodischen Vorgehensweise unterstützt werden. Um die Komplexität der beteiligten Elemente und des dezentralen Entwicklungsprozesses zu beherrschen ist ein Versions-, Konfigurations- und Variantenmanagement für die Modelle und die zugehörigen Parametersätze der CAE[8]-Werkzeuge erforderlich.

## 1.1 Motivation und Zielsetzung

Die systematische Integration mechatronischer Systeme nimmt in der Automobilbranche einen immer wichtigeren Stellenwert ein, um den steigenden Anforderungen der Kunden und gesetzlichen Vorgaben entgegen zu treten. Durch zusätzliche neue Funktionen und die Optimierung vorhandener erhöht sich die Komplexität der im Automobil verbauten Steuergeräte. Weiterhin werden vor dem Hintergrund ständig zunehmender Variantenvielfalt, der zusätzlichen Besetzung von Nischensegmenten und stetig verkürzter Produktzyklen eine steigende Produktqualität und kurze Entwicklungszyklen gefordert. Um dieser Entwicklung Rechnung zu tragen und in der Lage zu sein, neue Regelalgorithmen (als Bestandteil mechatronischer Fahrzeugsysteme) effektiv und in kurzer Zeit entwickeln zu können, ist eine integrative Entwicklungsmethode gefordert. Diese soll einerseits den modellbasierten Ansatz beim Entwurf mechatronischer Systeme unterstützen und somit andererseits ein rechnergestütztes methodisches Vorgehen ermöglichen. Die dabei anzuwendende integrative Entwicklungsmethode unterstützt einen Test der Steuergerätefunktionen durch Simulation in einer frühen Phase des Entwicklungszyklus. Das daraus resultierende Potential wird durch diverse Studien der NASA[9] [Dab04], [Erl07] und Pfeifer [Pfe96] unterstrichen, in denen der Zusammenhang der entdeckten Fehler während des Produktentwicklungs- und Produktlebenszyklus in Abhängigkeit zum resultierenden Kostenfaktor für die Fehlerbehebung dargestellt wird (vgl. Abbildung 1.1).

[7] OEM: Unter einem eng. „Original Equipment Manufacturer“ wird in der Automobilindustrie ein Hersteller fertiger Produkte verstanden, die er unter eigenem Namen in den Handel bringt.

[8] Computer Aided Engineering (CAE) bezeichnet die rechnergestützte Analyse und Simulation physikalischer Systeme oder Vorgänge.

[9] NASA (National Aeronautics and Space Administration) ist die zivile US-Bundesbehörde für Luft- und Raumfahrt.

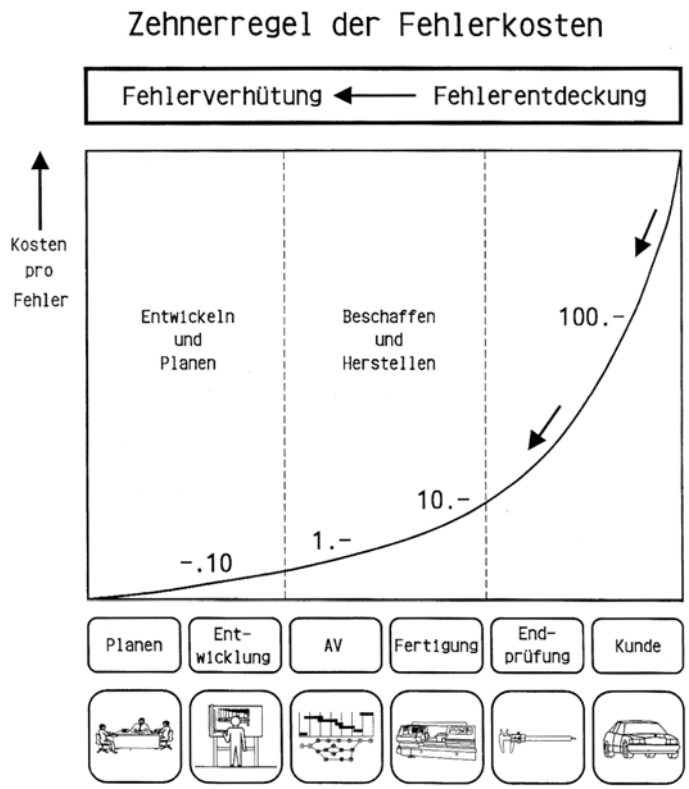


**Abbildung 1.1 - Zehnerregel der Fehlerkosten [Pfe96]**

Im Gegensatz zur herkömmlichen „Trial & Error"- Methode, die typischerweise erst nach der vollständigen Entwicklung des Steuergerätes zum Einsatz kommt, lassen sich mit modellgestützten Funktionstests Absicherungen bereits in einer frühen Entwicklungsphase durch Simulation im breiten Rahmen durchführen und ermöglichen so eine höhere Produktreife vor der realen Erprobung. Ein frühzeitiger Test und das Zusammenführen der heterogenen Entwicklungskomponenten sind vor dem Hintergrund reduzierter Entwicklungsbudgets, verkürzter Entwicklungszyklen und einer steigenden Variantenvielfalt unabdingbar. Insbesondere der frühzeitige Test von Steuergerätefunktionen, die Teil mechatronischer Fahrzeugsysteme sind, erhalten dadurch einen besonderen Stellenwert im Entwicklungsprozess. Beginnend mit Model-in-the-Loop[10], Software-in-the-Loop[11] bis hin zu Hardware-in-the-Loop[12] kann in einem durchgängigen Prozess getestet werden und somit eine höhere Reife der Steuergerätefunktion vor dem Einsatz im realen Fahrzeug er-

[10] Modell-in-the-Loop-Simulation (MiL) bezeichnet eine Simulation/Tests der Regelstrecke und des Reglers auf einem Computer. Der Regler liegt als Algorithmus vor. MiL-Tests werden häufig für Konzeptuntersuchungen eingesetzt.

[11] Unter Software-in-the-Loop-Simulation (SiL) wird die Simulation des Reglers als Seriencode mit einem Modell der Reglerstrecke zum Test verstanden. Typischerweise werden mit SiL-Tests Modultests durchgeführt.

[12] Tests während einer Hardware-in-the-Loop-Simulation (HiL) werden mit einem simulierten Modell der Reglerstrecke und einem in Hardware vorliegenden Steuergerät durchgeführt. Typischerweise werden hier Systemtests in Echtzeit durchgeführt.

reicht werden. Basis für den frühzeitigen Test sind Streckenmodelle[13], mit denen ein breiter Simulationsumfang abgedeckt werden kann, um die gewünschte Regelfunktion und die geforderte Regelgüte sicherstellen zu können. Ein wesentlicher Bestandteil für die Entwicklung von Steuergerätefunktionen ist die Abbildung des Streckenverhaltens. Eine effiziente Entwicklung setzt dabei eine Modularisierung als Vorstufe für die Wiederverwendung sowie die Erweiterung vorhandener Elemente voraus.

Um in der heterogenen Entwicklungslandschaft einen integrativen Entwicklungsprozess mechatronischer Fahrzeugsysteme zu gewährleisten sowie eine Mehrfachentwicklung oder einen Mehrfacheinkauf durch verschiedene organisatorische Einheiten zu verhindern, ist eine datenbankbasierte Softwareumgebung als Integrationsplattform für Simulationsmodelle erforderlich. Mit Hilfe dieser sollen folgende Ziele erreicht werden:

- Ein zentrales Versions-, Konfigurations- und Variantenmanagement für alle Funktions- und Streckenmodelle schafft mehr Transparenz zwischen den Entwicklern. Im Sinne eines integrativen Entwicklungsprozesses ist dies die Grundlage für einen frühzeitigen, ortsungebundenen und zeitunabhängigen Ergebnisaustausch und -abgleich.
- Ein paralleles Arbeiten in Form von Softwarekooperationen zwischen dezentralen Entwicklern mit einem kontinuierlichen Informationsfluss im Falle der Weiterentwicklung wird ermöglicht.
- Die effiziente Erstellung neuer Konfigurationen durch Wiederverwendung von vorhandenen Modellen, wenn erforderlich mit der Abbildung von Datenvarianten.
- Die durchgängige Verwendung von Modellen in den Ebenen Model-, Software- und Hardware-in-the-Loop wird gewährleistet.
- Eine datenbankbasierte Softwareumgebung für Simulationsmodelle ist zudem die Voraussetzung für eine Qualitätssteigerung durch einen gelebten durchgängigen Absicherungsprozess mit Model-, Software- und Hardware-in-the-Loop und unterstützt diesen Prozess in entscheidendem Maße.
- Interaktive Einbindung der Softwareumgebung in die domänenspezifischen CAE-Werkzeuge zum Aufbau von Modellen in der vom Entwickler gewohnten Umgebung.

In der vorliegenden Arbeit wird eine solche interaktive Softwareumgebung zur Unterstützung der Entwicklung mechatronischer Fahrzeugsysteme und Funktionsabsicherung von Steuergerätefunktionen mit virtuellen Prototypen konzipiert und realisiert. Diese Softwareumgebung benötigt ein Datenbanksystem im Hintergrund und wird den Grundbaustein zur Erfüllung der oben genannten Ziele darstellen.

[13] Das Streckenmodell ist ein Synonym zu dem Begriff der „Regelstrecke“ in der Regelungstechnik. Abgebildet wird durch das Streckenmodell das physikalische Verhalten des zu regelnden Elements. Kontextsensitiv werden darin mehrere verschiedene Teilmodelle zusammengefasst. In der Fahrdynamik gehört dazu typischerweise Fahrzeugmodell, Lenkung, Motor und Antriebsstrang etc.

## 1.2 Struktur der Arbeit

In Kapitel 2, dem „Stand der Technik“, wird zunächst auf die grundlegenden Konzepte der Entwicklung mechatronischer Fahrzeugsysteme eingegangen. Anschließend werden die bekannten Ansätze zur modellbasierten Entwicklung mit einem zentralen Datenbanksystem und lokalen Modelldatenbanken dargestellt. Das Kapitel 2 endet mit einem Fazit zu den zuvor betrachteten Werkzeugen und unterstreicht den Bedarf an einer Softwareumgebung als Integrationsplattform zur Entwicklung mechatronischer Fahrzeugsysteme mit einem zentralen Datenbanksystem zur strukturierten Ablage von Modellen und Parametern der während der Entwicklung benötigten Komponenten. Basierend auf den in Kapitel 2 herausgearbeiteten Informationen erfolgt in Kapitel 3 das Zusammenstellen der Anforderungen für generische und mechatronische Modelle im Kontext von Fahrzeugsystemen. Die in Abschnitt 3.2 aufgestellte Makroarchitektur des Konzeptes der neuen Softwareumgebung mit dem Namen „MeMo“ zur Entwicklung mechatronischer Fahrzeugsysteme wird danach mit konkreten Anwendungsfällen für die Unterstützung der Entwickler belegt. Die Anwendungsfälle unterscheiden sich bezüglich der Tätigkeitsfelder (Rollen) im Entwicklungsprozess mechatronischer Fahrzeugsysteme. In Kapitel 4 erfolgt die Festlegung der einzelnen Komponenten im Makrokonzept der Softwareumgebung. Im Zuge dessen werden gängige Datenbanksysteme analysiert und eine prototypische Entwicklungsumgebung für die interaktive Anbindung der neu entstehenden Softwareumgebung ausgewählt. Kapitel 5 beschreibt detailliert die implementierten Algorithmen der Softwareumgebung MeMo und die daraus resultierende Unterstützung für die Entwickler mechatronischer Fahrzeugsysteme. Die Bereitstellung der geschaffenen Softwareumgebung wird in Kapitel 6 erläutert und an einem Beispiel die Applikation im Entwicklungsprozess dargestellt. Abschließend wird in Kapitel 7 eine Zusammenfassung der vorliegenden Arbeit gegeben, bevor ein Ausblick mit zukünftigen Anwendungsbereichen und Funktionalitäten folgender Entwicklungsstufen vorgeschlagen wird.

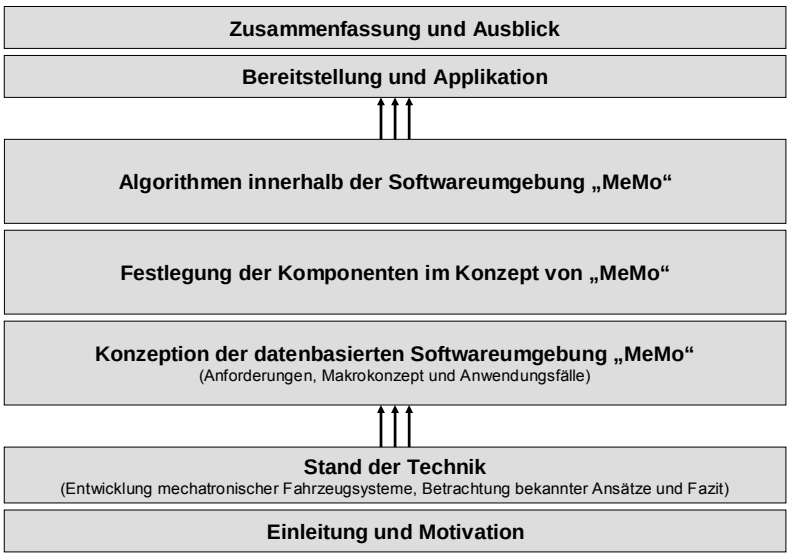


**Abbildung 1.2 - Aufbau der vorliegenden Arbeit**

# 2 Stand der Technik

Ziel der vorliegenden Arbeit ist es, die modellbasierte Entwicklung und Funktionsabsicherung mechatronischer Fahrzeugsysteme zu unterstützen. Insbesondere die speziellen Anforderungen durch mechatronische Fahrzeugsysteme sollen dabei berücksichtigt und in einer neuen Softwareumgebung für eine integrative Entwicklung mechatronischer Fahrzeugsysteme umgesetzt werden. In Abschnitt 2.1 wird kurz auf die Charakteristik mechatronischer Systeme im Allgemeinen und die ergänzenden Rahmenbedingungen bei Automobilherstellern eingegangen.
Das Thema Datenbank für Entwicklungsfragmente ist heute insbesondere aus dem Themenkomplex Produktdatenmanagement (PDM) bekannt. Während der PDM-Einsatz im Zusammenspiel mit CAD-Systemen bereits recht gut funktioniert sind andere, ebenfalls im Konstruktions- und Entwicklungsprozess eingesetzte Systeme wie z. B. Simulationswerkzeuge, nur in den seltensten Fällen in die PDM-Umgebung integriert [Vet03]. Diese fordern in der Regel einen anderen Umgang mit den Simulationsmodellen, was bei verschiedenen Automobilherstellern und Zulieferern zu einer hausinternen Lösung geführt hat. Diese aus der Literatur bekannten Systeme bei Automobilherstellern und -zulieferern, die zeitgleich zur vorliegenden Arbeit konzipiert und teilweise implementiert wurden, werden im Folgenden analysiert. Im Fokus der Softwaresysteme steht typischerweise die Unterstützung der modellbasierten Funktionsentwicklung und -absicherung.
Abschließend wird ein Fazit zur Beurteilung der vorhandenen Systeme gezogen und damit aufgezeigt, in welchem Maße die Entwicklung mechatronischer Fahrzeugsysteme durch diese unterstützt werden kann. Des Weiteren werden die Defizite dargestellt, die zugleich einen Teil der Basis für die Anforderungen (vgl. Abschnitt 3.1) der neuen datenbankbasierten Integrationsplattform „MeMo“ bilden.

## 2.1 Mechatronische Fahrzeugsysteme

Kern eines modernen Verständnisses von Mechatronik ist die Integration unterschiedlicher Teilmodelle zu einem Ganzen. Die interdisziplinäre Kooperation zwischen Entwicklern bringt dabei synergetische Effekte durch das Zusammenführen verschiedener Technologien, wobei nicht die Einzeltechnologie, sondern die Kombination von Technologien ein Optimum darstellt [Gau00] [Rot02]. Die Integration der Technologien muss möglichst früh in der Entwicklung, während der Spezifikations- und Konzeptphase, erfolgen und setzt eine ganzheitliche Betrachtung des Systems sowie interdisziplinäres Denken der Entwickler voraus. Mechatronik stellt in diesem Kontext mehr eine integrative und interdisziplinäre Strategie als eine Technologie dar. Dabei bildet die ganzheitliche Betrachtung aller beteiligten Systemkomponenten die Grundlage für den mechatronischen Entwicklungsprozess.

Im Kraftfahrzeug werden heute zunehmend mehr mechatronische Systeme wie beispielsweise EPS[14], ESP[15] oder ACC[16] eingesetzt, um den steigenden Anforderungen an Fahrsicherheit und Fahrkomfort begegnen zu können. Diese sind typischerweise sehr komplex und heterogen aufgebaut. Unterschiedliche Domänen, die von unterschiedlichen Entwicklungsteams bzw. von unterschiedlichen Entwicklern bearbeitet werden, werden fast ausschließlich verteilt entwickelt und in einem übergeordneten System, dem sogenannten „Integrationssystem“, zusammengeführt.
In den nachstehenden Abschnitten wird die Entwicklung, Strukturierung und Funktionsabsicherung mechatronischer Fahrzeugsysteme kurz betrachtet.

### 2.1.1 Entwicklung mechatronischer Fahrzeugsysteme

Grundlage zur Erläuterung der Entwicklung mechatronischer Fahrzeugsysteme stellt die VDI-Richtlinie 2206 [VDI06] mit dem darin abgebildeten V-Modell als Makrozyklus für die Entwicklung dar. Die Erzeugung von serienreifen Produkten erfordert ein mehrfaches Durchlaufen des in Abbildung 2.1 dargestellten V-Modells. Darin enthalten sind auch Zwischenprodukte wie Labormuster[17], Funktionsmuster[18] und Vorserienprodukte[19], durch die beim Durchschreiten des V-Modells allmählich ein höherer Reifegrad des Produktes erreicht wird. Eine grobe Gliederung wird in die nachstehend beschriebenen Phasen vorgenommen [VDI06].

[14] Elektrische Servolenkung (eng. Electrical Power Steering): Durch das System wird die mechanische Verbindung zwischen dem Lenkrad und den Rädern erhalten. Durch einen parallel zum Handmoment wirkend Elektromotor (EPS) lassen sich über elektrische Eingänge zusätzliche Funktionen, z. B. zum automatischen Parken, realisieren [Ise06], [Bos07].

[15] Elektronisches Stabilitätsprogramm (eng. Electronical Stability Program): Das System findet Einsatz, um die Schleuderbewegungen durch Giermomentenreduzierung über das Bremsen einzelner Räder zu dämpfen und damit das Fahrzeug auf Kurs zu halten [Ise06], [Bos07].

[16] Adaptive Cruise Control: Das System wird mit Radarsensorik zur Abstands- und Geschwindigkeitsregelung eingesetzt [Ise06], [Bos07].

[17] Labormuster: Erste Wirkprinzipien und Lösungselemente, Grobdimensionierung, erste Funktionsuntersuchungen [Ise06].

[18] Funktionsmuster: Weiterentwicklung, Feindimensionierung, Integration verteilter Komponenten, Leistungsmessung, Standard-Schnittstellen [Ise06].

[19] Vorserienprodukt: Berücksichtigung der Fertigungstechnik, weitere modulare Integrationsstufen, Kapselung, Feldtests [Ise06].

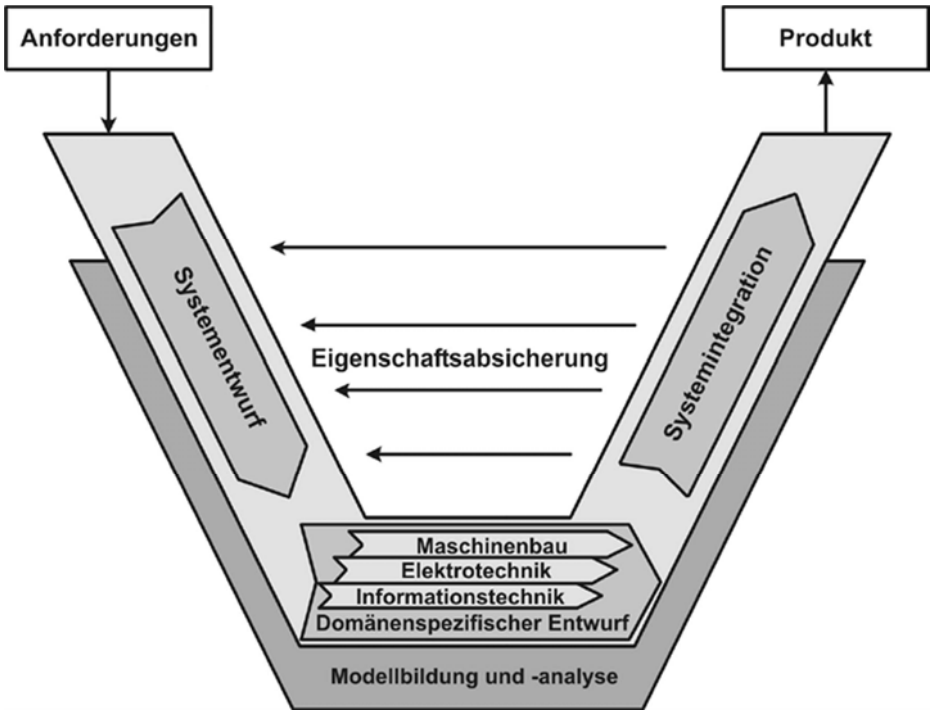


**Abbildung 2.1 - V-Modell mit Darstellung der Funktionsabsicherung durch Simulation [VDI06]**

- Anforderungen: Die Aufgabenstellung wird in Form von Anforderungen beschrieben. Diese bilden zugleich den Maßstab, anhand dessen das spätere System bewertet wird.
- Systementwurf: Ein domänenübergreifendes Lösungskonzept wird festgelegt, das die wesentlichen physikalischen und logischen Wirkungsweisen des zukünftigen Produktes beschreibt. Hierzu wird die Gesamtfunktion eines Systems in wesentliche Teilfunktionen zerlegt. Diesen werden die geeigneten Wirkprinzipien bzw. Lösungselemente zugeordnet und die Funktionserfüllung im Systemzusammenhang geprüft.
- Domänenspezifischer Entwurf: Es erfolgt eine weitere Konkretisierung des im Systementwurf erarbeiteten Lösungskonzepts in den beteiligten Domänen. Detailliertere Auslegungen und Berechnungen sind notwendig, um insbesondere bei kritischen Funktionen die Funktionserfüllung sicherzustellen.
- Systemintegration: Ziel ist es, hier das Zusammenwirken der Ergebnisse aus den einzelnen Domänen in einem Gesamtsystem zu untersuchen.
- Eigenschaftsabsicherung[20]: Der Fortschritt aus dem domänenspezifischen Entwurf muss fortlaufend anhand des spezifischen Lösungskonzeptes und den Anforderungen überprüft werden. Sicherzustellen ist, dass die tatsächlichen mit den gewünschten Systemeigenschaften übereinstimmen. Hierzu zählt unter anderem die

[20] Die Phase der Eigenschaftsabsicherung wird in der Literatur (z. B. [Alb06]) auch mit „Verifikation und Validierung“ bezeichnet.

frühzeitige Simulation des Systems durch die Verfahren MiL, SiL und HiL, die in Abschnitt 2.1.3 im Detail beschrieben werden.

- Modellbildung und -analyse: Die beschriebenen Phasen werden flankiert durch die Abbildung und Untersuchung der Systemeigenschaften mit Hilfe von Modellen und rechnergestützten Werkzeugen.
- Produkt: Ergebnis des durchlaufenen Makrozyklus ist das fertige Produkt oder Modul bzw. die Baugruppe oder ein Teilelement, das sowohl als real existierendes Erzeugnis, aber auch als zunehmende Konkretisierung des zukünftigen Produktes (z. B.: Labormuster, Funktionsmuster, Vorserienprodukt) verstanden werden kann.

Die im Rahmen dieser Arbeit konzipierte und realisierte Softwareumgebung unterstützt die modellbasierte Entwicklung (Phase: Modellbildung und -analyse) mechatronischer Fahrzeugsysteme und die damit verbundenen Verfahren zur virtuellen Absicherung (Phase: Eigenschaftsabsicherung) maßgeblich.

### 2.1.2 Strukturierung mechatronischer Fahrzeugsysteme

Wie zuvor in den unterschiedlichen Phasen beschrieben, erfordert die Entwicklung mechatronischer Fahrzeugsysteme einerseits Spezialisten für den domänenspezifischen Entwurf der Systeme, andererseits ist nur mit einer ganzheitlichen Betrachtung und dem entsprechenden rechnergestützten Methodeneinsatz ein optimal kontrolliertes Systemverhalten erreichbar. Ein wichtiger Aspekt ist eine strukturierte Herangehensweise, Hierarchisierung bzw. Modularisierung mechatronischer Systeme, um in der frühen Entwicklungsphase einen integrativen Entwicklungsprozess optimal zu unterstützen. Modularisierung bedeutet die Kapselung von Funktionalitäten des gesamten Systems in bestimmte Module. Die Hierarchisierung hingegen beschreibt die Zusammenfassung von Modulen in höherwertige Gruppen inklusive einer Schnittstellenspezifikation [Liu05D]. In der Praxis bedeutet Modularisierung eine Zerlegung des gesamten mechatronischen Fahrzeugsystems mittels „top-down-design“[21] in einzelne autarke „mechatronische Funktionsmodule“. Diese sind hierarchisch so aufgeteilt, dass eine Kommunikation mit der Umgebung oder anderen Funktionsmodulen über eine möglichst schmale und definierte Schnittstelle erfolgen kann. Für eine Modularisierung/Hierarchisierung werden vier Ebenen eingeführt [Hon97], [Liu05D]:

- Mechatronisches Funktionsmodul (MFM): Das MFM ist die kleinste mechatronische Einheit, bestehend aus den Teildisziplinen mechatronischer Systeme (mechanische Tragstruktur, Sensor, lokale Informationsverarbeitung und Aktor). Die implementierte Informationsverarbeitung ist mittels Sensoren und Aktoren mit der mechanischen Tragstruktur verbunden und wird für konventionelle Regelungsaufgaben eingesetzt. Das Funktionsmodul wird als lokales, austauschbares Element verstanden, das eindeutig definierte physikalische und informations-

[21] Die schrittweise Verfeinerung der Grobstruktur in genauer spezifizierte Strukturelemente wird als „top-down-design“ bezeichnet [VDI06].

technische Schnittstellen zu anderen MFM oder übergeordneten Elementen besitzt.

- Mechatronische Funktionsgruppe (MFG): Die MFG gruppiert eine oder mehrere MFM. Die Funktionsgruppe besitzt Sensorik und Informationstechnik, jedoch keine mechanische Tragstruktur oder Aktorik. Eine Erfüllung ihrer Funktion kann nur durch die unterlagerten MFM und der dort vorhandene Aktorik stattfinden. In den meisten Fällen erfüllt die MFG einen Teil der Informationsverarbeitung und dient vorwiegend zur Strukturierung der Informationsverarbeitung und der mechatronischen Funktionsmodule. Beispiele für MFMs sind ESP, ABS etc.
- Autonomes mechatronisches System (AMS): Fasst zwei oder mehrere MFG und MFM zusammen, die sowohl physikalisch und/oder informationstechnisch gekoppelt sind. In der Regel besteht über die mechanische Struktur eine passive Verbindung zwischen den gruppierten Elementen. Die Informationsverarbeitung des AMS steuert übergeordnet die hierarchisch gegliederten MFGs und MFMs. Das AMS stellt ein vollständiges und autonomes mechatronisches System dar, das sich selbst mit Energie vorsorgt und über keine eigenen Sensoren und Aktoren verfügt. Als globale Informations- und Datendrehscheibe für das autonome mechatronische System werden informationstechnische Schnittstellen zu weiteren Systemen bereitgestellt.
- Vernetztes mechatronisches System (VMS): Ein VMS bezeichnet die oberste Ebene und kombiniert mehrere AMS auf rein informationstechnischer Basis. Die Informationsverarbeitung wird durch diskrete Ereignisse dominiert. Das VMS entscheidet aufgrund der von dem AMS bereitgestellten Informationen und koordiniert diese. Die Anzahl der beteiligen AMS kann sich dynamisch verändern, wie das Beispiel Car-2-Car-Kommunikation[22] zeigt.

In der Praxis hat sich die oben dargestellte Strukturierung mechatronischer Systeme vielfach bewährt. Abbildung 2.2 zeigt beispielhaft eine hierarchische Strukturierung eines mechatronischen Fahrzeugsystems.

[22] Car-2-Car-Kommunikation bezeichnet den Datenaustausch zwischen zwei oder mehreren Kraftfahrzeugen. Ziel ist es, den Fahrer frühzeitig auf kritische Situationen hinzuweisen.

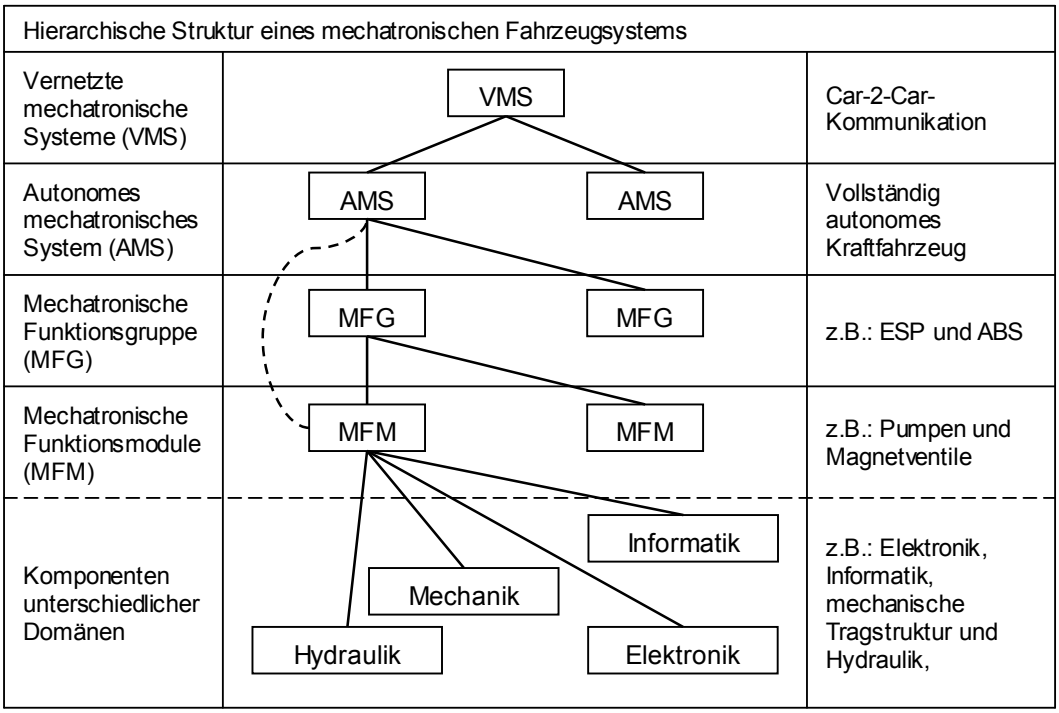


**Abbildung 2.2 - Beispielhafte Strukturierung eines mechatronischen Fahrzeugsystems**

Abbildung 2.2 macht deutlich, dass eine Modularisierung in einzelne Module unumgänglich ist, um die Komplexität mechatronischer Fahrzeugsysteme zu beherrschen. Diese schafft die Voraussetzung, mechatronische Fahrzeugsysteme im Team mit mehreren Entwicklern in einem „bottom-up-design“[23] bearbeiten zu können und ermöglicht eine Wiederverwendung bereits vorhandener Komponenten zum Aufbau neuer Strukturen in kurzer Zeit.

Um von diesen Vorteilen zu profitieren, sind Algorithmen bzw. eine neue Softwareumgebung erforderlich, die ein Arbeiten von mehreren Entwicklern in Form von Softwarekooperationen ermöglicht und somit die notwendige Transparenz zwischen den beteiligten Entwicklern aus den unterschiedlichen Domänen mechatronischer Komponenten schafft. Weiterhin sind Algorithmen erforderlich, die eine eindeutige hierarchische Zuordnung der Module bzw. Teilsysteme zueinander gestatten und so den Einsatz in ähnlichen Fahrzeugprojekten erlauben. Um dies zu gewährleisten ist ein zentrales Datenbanksystem für den Zugriff durch alle im Entwicklungsprozess beteiligten Entwickler unerlässlich. Idealerweise ist dieses nach den Merkmalen mechatronischer Fahrzeugsysteme strukturiert und implementiert zusätzlich Methoden für eine Versions- und Konfigurationskontrolle zum Einsatz in diversen Fahrzeugprojekten als ein Fundament.

[23] Das Vorgehen, getrennt entwickelte und optimierte Baugruppen zu einem Gesamtsystem zusammen zu führen, wird als „bottom-up-design“ bezeichnet [VDI06].

### 2.1.3 Funktionsabsicherung mechatronischer Fahrzeugsysteme

Im Fokus der Funktionsabsicherung steht der Regelalgorithmus, der durch Simulation in den drei Ebenen Model-, Software- und Hardware-in-the-loop getestet wird. Wie in Abschnitt 2.1.1 beschrieben, ist dieser ein Teil der Phase „Eigenschaftsabsicherung“ und geht im Rahmen der modellbasierten Entwicklung mit der „Modellbildung und -analyse“ einher. Abbildung 2.3 zeigt die Methode mit den aufeinander aufbauenden Ebenen der Funktionsabsicherung.

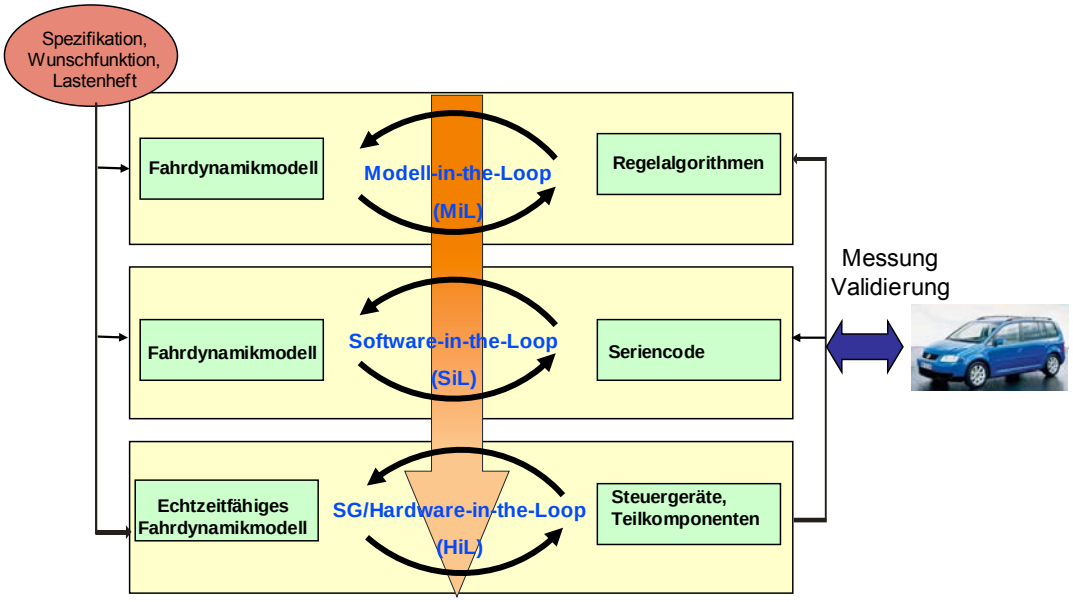


**Abbildung 2.3 - Modellgestützter Funktionsentwicklungs- und Absicherungsprozess [Liu05]**

Die Grundlage für die Funktionsabsicherung des Regelalgorithmus stellt die in der „Anforderungs“-Phase festgelegte Wunschfunktion bzw. Spezifikation dar. Basierend auf dieser erfolgt eine Bewertung des Systemverhaltens auf allen drei Ebenen.
In allen drei Ebenen Model-, Software- und Hardware-in-the-Loop wird die Regelfunktion über die im Vorfeld definierten Schnittstellen in die zu regelnde Strecke eingebettet und gemäß der vorgegebenen Spezifikation getestet. Dabei wird stets auf die vorherige Ebene zurückgegriffen, um Fehlerquellen zu beseitigen und die gewünschten Funktionen zu erzielen.
Zentrales Element der dargestellten Methoden zur Funktionsabsicherung ist das Streckenmodell, das in Abbildung 2.3 beispielhaft als Fahrdynamikmodell abgebildet ist und zusammen mit dem Testgegenstand simuliert wird. Das Streckenmodell wird in dem CAE-Werkzeug aufgebaut und anhand realer Messungen validiert. Ziel ist es, das Streckenmodell möglichst durchgehend in der dargestellten Simulationsmethode für Entwicklung und Test komplexer mechatronischer Systeme zu verwenden, um diesen beherrschbar und reproduzierbar zu halten. Die Fehlerquote im Produkt kann dadurch wesentlich minimiert und somit die Entwicklungszeit verkürzt werden [Liu05].
In den beiden Ebenen Model-in-the-Loop und Software-in-the-Loop wird der Regelalgorithmus bzw. der Seriencode in die Simulation eingebunden. Sowohl Regelalgorithmus als

auch Seriencode liegen ausschließlich als Software für die Funktionsabsicherung vor. In der Ebene Hardware-in-the-Loop liegt der Testgegenstand als reales Steuergerät mit darauf implementiertem Reglercode vor. Das Streckenmodell muss dafür die Signale für das Steuergerät in Echtzeit[24] bereitstellen. Aufgrund der rechenintensiven und teilweise komplizierten Modellierung von Bauteilen des Streckenmodells werden partiell zusätzliche Teilkomponenten (z. B. Drosselklappe) als reale Bauteile in die Simulation eingebunden.

**Model-in-the-Loop (MiL)**

Unter Model-in-the-Loop-Simulation versteht man die Einbettung des Regelalgorithmus in eine Simulationsumgebung (sog. Streckenmodell oder Reglerstrecke). Üblicherweise steht der Regelalgorithmus als editierbarer Algorithmus bzw. Code (z. B. Blockschaltbild oder Zustandsautomat) zur Verfügung und kann im geschlossenen Regelkreis durch die Simulation auf einem Computer untersucht werden. MiL-Tests werden häufig für Konzeptuntersuchungen eingesetzt. Ziel ist es, in dieser Phase das Potential des Systems zu analysieren und das qualitative Verhalten gegenüber der Spezifikation zu vergleichen, um damit die vorgegebenen Gesamtfahrzeugziele bzgl. Fahrdynamik, Sicherheit und Komfort zu validieren.

**Software-in-the-Loop (SiL)**

Nachdem der Funktionsalgorithmus die im Lastenheft geforderte Spezifikation erfüllt, erfolgt der Übergang in die SiL-Ebene. Der Regelalgorithmus ist mittlerweile in kompilierten Code überführt worden. Oftmals und insbesondere dann, wenn der Regelalgorithmus dezentral (z. B. beim Zulieferer) entwickelt wurde, erfolgt in dieser Phase ein erneuter Vergleich der Ergebnisse mit der Spezifikation bzw. dem Lastenheft für die geforderte Funktion. SiL-Tests werden häufig für Modultests eingesetzt und auf einem Computer durchgeführt. Ziel ist die Absicherung des generierten Codes, um somit eine Steigerung der Softwarequalität für die nachfolgende Codegenerierung auf dem Steuergerät zu erzielen.

**Hardware-in-the-Loop (HiL)**

In der Ebene HiL wird der Regelalgorithmus bereits als Funktion auf einem realen Steuergerät ausgeführt. Dieses wird oftmals inklusiv realer Teilkomponenten in die virtuelle Simulation des Streckenmodells zur Beurteilung eingebunden. Die Simulation dynamischer Systeme durch physikalische und mathematische Modelle muss dabei in Echtzeit und beispielsweise unter Nachbildung der resistiven, induktiven und kapazitiven Lasten für die Steuergerätefunktion erfolgen. Hierzu wird ein Simulator mit echtzeitfähigem Betriebssystem mit dem realen Steuergerät für die Simulation gekoppelt.

[24] Von Echtzeit wird gesprochen, wenn das Streckenmodell in der Lage ist, rechtzeitig die Ausgangswerte des Steuergerätes zu verarbeiten und innerhalb eines zuvor definierten Zeitrasters die neu berechneten Eingangswerte für das Steuergerät bereitstellt. Daraus resultieren eventuell Kompromisse bezüglich der Genauigkeit der verwendeten Komponentenmodelle, da beispielsweise die Abbildung von komplexen Reifen oder Hydraulikmodellen einen hohen Rechenaufwand erfordern [Ise06].

Durchgeführt werden Systemtests[25], die beispielsweise vernetzte Funktionen (z. B. ESPII) über mehrere Steuergeräte hinweg und den Test der Sicherheitsfunktion (z. B. Fail-Safe) des jeweiligen Steuergerätes beinhalten. Weiterhin werden die Testergebnisse der Ebenen MiL/SiL in einer Echtzeitumgebung weitestgehend validiert und verbessert.

Die dargestellte Methode zur Funktionsabsicherung wird wie in Abbildung 2.1 parallel zum Entwicklungsprozess des mechatronischen Fahrzeugsystems angewandt. Dadurch, dass mehrere Entwickler an den Systemen unterschiedlicher Domänen arbeiten, ist ein Entwicklungsprozess in Form von Softwarekooperationen, der eine Kooperation und Kommunikation zwischen den beteiligten Entwicklern und Testern schafft, unerlässlich. Für die Durchführung der Funktionsabsicherung auf den unterschiedlichen Ebenen sind Konfigurationen zu erzeugen und so festzuhalten, dass die Ergebnisse jederzeit reproduziert und nachvollzogen werden können.
Seitens einer Softwareumgebung zur Unterstützung der hausinternen Entwickler ist insbesondere die Modellbildung der für die Simulation benötigten Streckenmodelle hervorzuheben, da üblicherweise der Regelalgorithmus bei Zulieferern implementiert wird. Die bereitgestellten Zwischenstände in Form des Regelalgorithmus, Seriencode oder Steuergerätes sind wie oben beschrieben zu testen und über ein zentrales Datenbanksystem für Entwickler in anderen Organisationseinheiten bereitzustellen.

## 2.2 Modellbasierte Entwicklung mit einer zentralen Datenbank

Nach der Betrachtung mechatronischer Systeme (Abschnitt 2.1) werden im Folgenden bei den Automobilherstellern und Zulieferern eingesetzten datenbankbasierten Softwareumgebungen für die modellbasierte Entwicklung analysiert. Eine tabellarische Übersicht der Werkzeuge ist im Anhang 9.1 abgebildet.

**Modelldatenbank für komplexe Simulinkmodelle (BMW)**

Aus den Veröffentlichungen [Sch05] und [Kva06] ist eine Modelldatenbank bekannt, die derzeit prototypisch in den Entwicklungsbereichen Antriebsstrang und Fahrwerk angewendet wird. Die Kernfunktionalität ist die Organisation von hierarchisch strukturierten Simulationsmodellen in Simulink und deren Versionierung in einem Datenbanksystem. Adressiert wird mit der Modelldatenbank die modellbasierte Funktionsentwicklung und -absicherung mittels MiL, SiL und HiL basierend auf Matlab/Simulink[26].
Das eingesetzte IT-System stellt eine zentrale Datenbank für den Austausch der Simulationsmodelle mit Versionsmanagement für den Entwickler bereit. In der Datenbankstruktur erfolgt die Unterteilung in generische Werkzeuge und Zubehör, Parameterdaten, modellspezifisches Zubehör und Modelle. Aus dieser Struktur werden sog. „Module“ oder „Pakete“ als Konfigurationen zusammengestellt. Diese können wiederum ineinander geschachtelt

[25] Beim Systemtest wird das System gegen die Spezifikation der logischen Systemarchitektur getestet [Schae03].

[26] Matlab/Simulink ist ein Softwaresystem zur blockschaltbildorientierten Modellierung von dynamischen Systemen. Vertrieben wird dieses durch die Firma *The MathWorks, Inc.*

werden. Die Konfiguration wird mit einer XML-Datei beschrieben, durch die einzelne Dateien in der Datenbank miteinander in Beziehung gesetzt werden und somit eine Dateigruppe repräsentieren. Die Konfiguration dient als Datenschnittstelle für den Datentransport zwischen der lokalen Festplatte und dem Datenbanksystem und für das Versionsmanagement der einzelnen Dateien. Die XML-Datei als Basis einer Konfiguration untersteht selbst einer Versionskontrolle und ermöglicht dem Anwender ein Variantenmanagement für Modellvarianten durch das Erzeugen eines neuen Zweiges, einem sog. „Branch", in der Datenbank. Basierend auf den XML-Dateien erfolgt vom Entwickler bzw. dem Anwender das Herunterladen der Dateien bzw. Modelle, die mit Hilfe der „Build-Tools" in die Entwicklungsumgebung eingebunden und simuliert werden können. Die „Build-Tools" stellen die lokal verfügbaren Dateien in Matlab zur Verfügung und öffnen die bestehenden Bibliotheken für die Anwendung. Weiterhin unterstützen die „Build-Tools" das Speichern eines hierarchisch gegliederten Modells in die Datenbank.
Eine Beschreibung der Modelle erfolgt auf Basis einer zusätzlichen Datei für die Modelldokumentation, welche als separate Datei in der Konfiguration enthalten ist. Zusatzinformationen in Form von Metadaten[27] wie Status, Release, Beschreibung des Modellursprungs, Richtlinien etc. sind auf der XML-Datei verfügbar. Eine Dokumentation von Änderungen auf Dateiebene ist nicht möglich.
Das bei der BMW AG entstandene Softwarewerkzeug integriert sich durch die „Build-Tools" sehr gut in die Entwicklungsumgebung Matlab/Simulink. Durch die fehlenden Metadaten auf Dateiebene ist eine Rückverfolgung von Änderungen nur durch das Öffnen der Dateien/Modelle möglich, sofern darin dokumentiert. Der geringe Satz von Metadaten auf einem Modul erschwert für den Anwender die Auswahl vorhandener Modelle. Ebenso lassen sich Datenvarianten nur über zusätzliche Verzweigungen eines Moduls abbilden, was eine Suche nach vorhandenen Parametersätzen auf ein bestehendes Modell sehr schwer gestaltet. Unklar ist, wie bei einer interaktiven Entwicklung durch mehrere Entwickler der lokale Status von Modulen/Dateien (z. B.: „Modul befindet sich in Entwicklung", „Aktuellere Version vorhanden" etc.) erkannt und Teilelelemente aktualisiert werden können. Das Berechtigungssystem unterscheidet generell zwischen Entwickler und Anwender; nicht berücksichtigt ist eine Berechtigungsstufe zwischen den Entwicklern.

**Umgebung für Softwareentwicklung im Automobil (Continental Teves)**

Für die Steuergeräte-Code-Entwicklung bei der Firma Continental Teves AG & Co. oHG wird eine zentrale Datenbank, die weltweit für die Entwicklungsstandorte zur Verfügung steht, genutzt. Diese wird in den Veröffentlichungen [Bun04] und [Cre04] vorgestellt.
Das im Hintergrund verwendete Datenbanksystem ist strukturiert nach den Bereichen „Project Codebasis" für die Funktionsalgorithmen und „Projekt Configuration" für den auftragsspezifischen Anteil, der durch Einstellungen und Parameter bestimmt wird. Die Konfiguration eines Projektes erfolgt mittels einer grafischen Oberfläche. Für die Anwendung wählt der Entwickler ein Projekt aus und startet somit den Transfer aller projektrelevanten

[27] Metadaten sind Informationen in Textform, die Markmale (sog. Attribute) beschreiben. Ein Beispiel hierfür ist der Änderungskommentar.

Dateien auf seine lokale Festplatte. Nach dem Herunterladen wird automatisch eine Header-Datei (*.h) generiert, welche `#define`-Anweisungen für die Konfiguration der Softwarealgorithmen enthält. Für das Arbeiten im Mehrbenutzerbetrieb sind Zugriffsmechanismen spezifiziert, welche das Sperren von Dateien bei Weiterentwicklung erlauben. Explizit wird darauf verwiesen, dass durch die zur Verfügung stehenden Mechanismen und das angebundene System hauptsächlich der einzelne Entwickler adressiert wird. Ein Zusammenarbeiten mehrerer Entwickler wird durch einen zusätzlichen Aufwand des Projektadministrators realisiert, was eine effektive Zusammenarbeit verhindert.
Ein Variantenmanagement wird in Matlab durch zwei Simulink-Blöcke „strap-start“ und „strap-end“ mit zwischengeschalteten Blöcken zur Parametrisierung konzipiert. Abhängig von der verwendeten Variante werden entsprechend die Parameterblöcke aktiviert. Das vorgestellte Konzept greift ausschließlich auf lokal verfügbare Parameterdateien zurück. Datenbankseitig werden keine Referenzen zur Abbildung der Datenvarianten zwischen Modell und Parametern im Sinne eines Datenvariantenmanagements dargestellt.
Es wird keine Aussage zu den Themen Dokumentation oder der Organisation hierarchisch gegliederter Modelle (Konfiguration) getroffen.
Für den Einsatz während der Entwicklung mechatronischer Fahrzeugsysteme fehlt unter anderem die Betrachtung von Streckenmodellen und übergeordneten Konfigurationen, die Referenzen zwischen Regelalgorithmus und Regelstrecke erlauben. Das Arbeiten in Softwarekooperationen mit einer transparenten Sicht des Anwenders auf den Status der beteiligten Elemente sowie eine unzureichende Abbildung von Datenvarianten auf Simulationsmodelle bzw. Algorithmen erschweren die Entwicklung mechatronischer Fahrzeugsysteme zusätzlich.

**DataDictionary (dSpace)**

Die Firma dSpace GmbH unterstützt die modellbasierte Funktionsentwicklung u.a. mit dem Produkt „TargetLink“. Das Softwareprodukt sowie das zugehörige „DataDictionary“ basieren auf Matlab/Simulink. Das „DataDictionary“ als zentrale Ablage für die Datenvarianten während der Funktionsentwicklung wird in den Veröffentlichungen [Bei06] und [Bei07] beschrieben.
Das „DataDictionary“ wird im Rahmen der Seriencodegenerierung mit „TargetLink“ eingesetzt und dient als Container, um die während der Funktionsentwicklung anfallenden Daten aufzunehmen. Als Daten werden Modell-Parameter (M-Dateien) und softwarebezogene Daten, wie sie typischerweise im Rahmen der Funktionsentwicklung und Code-Generierung für Simulink-Modelle auftreten, verstanden. Dies beinhaltet Variablen-, Typ- und Skalierungsobjekte sowie Betriebssystemobjekte wie Tasks, Events, Ressourcen und Schutzmechanismen. Das Ablegen von Funktionsmodellen selbst sowie Zusatzdaten (Referenzergebnisse etc.) ist nicht vorgesehen, da eine Ablage der Objekte nicht dateibasiert ist.
Eine Dokumentation der Objekte erfolgt direkt auf der grafischen Oberfläche des „DataDictionary“ mit fest definierten, aber obligatorischen Attributen wie beispielsweise Datentyp oder Skalierung. Die im „DataDictionary“ abgelegten Objekte werden als Referenz zu einem Funktionsalgorithmus gespeichert und ermöglichen dafür das Variantemanagement. Für die Anwendung der Daten wird zwischen zwei Rollen unterschieden: Der Administra-

tor kann für die Spezifikation / Parametrisierung Objekte anlegen, auf die der Anwender nur lesend Zugriff hat. Es existiert kein Nutzer- oder detailliertes Rollenkonzept für die Anwender der Software.
Über Im- und Exportmöglichkeiten kann der Anwender Daten in/aus Dateien (M, XML, ASAP etc.) schreiben oder lesen.
Das „DataDictionary“ ist nicht als firmenweites zentrales Datenbanksystem entwickelt worden, sondern fokussiert ausschließlich auf die Daten- bzw. Variantenorganisation von Funktionsmodellen für die Seriencodegenerierung, die lokal auf dem Computer des Anwenders vorhanden sind. Eine Verwendung für die zentrale Datenhaltung mechatronischer Teilelemente oder die modellbasierte Entwicklung wird dadurch und durch die fehlende Versions- und Konfigurationsverwaltung sowie den nicht gegebenen Mehrbenutzerbetrieb verhindert.

**Modellbasierte Entwicklung (Siemens VDO Powertrain)**

Ein weiteres hausinternes datenbankbasiertes Entwicklungswerkzeug wurde bei Siemens VDO AG entwickelt und wird dort derzeit für Systeme in der Antriebsstrangentwicklung eingesetzt [Kun04]. Bestehend auf den Einzelkomponenten „SDA Core Development“, „Model Based Testing“, „Rapid Prototyping“, „Automatic Code Generation“ und „Model Based Documentation“ stellt die Komponente „SDA - Core Development“ die Basis des Entwicklungswerkzeuges dar. Durch diese wird für den Entwickler ein „automotive Blockset“, der Modellierungsstandard und die Schnittstelle zu dem Konfigurationsmanagement bereitgestellt. Weitere Schnittstellen zu zentralen Datenbanken, z. B. „Requirement Engineering“, „Test Environment“ und „DataDictionary“, werden oder sollen zukünftig durch die anderen Komponenten hergestellt werden.
Für die Interaktion (Umgang mit Pfaden und Bibliotheken) mit der Entwicklungsumgebung Matlab/Simulink wird das lokale Dateisystem des Anwenders genutzt. In einer statisch definierten Verzeichnisstruktur, dem sogenannten „Matlab Development Space“ (MDS) werden ihm die Dateien für die Projektarbeit zur Verfügung gestellt. Der MDS bietet die Möglichkeit mehrere Projekte parallel abzulegen und diese separat in Matlab zu nutzen. Die Nutzung erfolgt über den „Matlab Integration Space“ (MIS), durch den die richtigen Pfade und Bibliotheken in Matlab zur Verfügung gestellt und direkt vom Anwender genutzt werden können. Die Bibliotheken enthalten alle generischen Bibliothekselemente, z. B. das „Automotive Blockset“, und zusätzlich die projektrelevanten Elemente.
Die beim Modellaufbau verwendeten Dateien bzw. referenzierten Dateien werden in einer XML-Datei festgehalten. Die Struktur der XML-Datei bildet die Konfiguration eines Modells ab und enthält die Dateinamen der referenzierten Elemente, den Speicherort in dem lokalen MDS sowie die Versionen der Dateien. Die generische XML-Datei ist unabhängig vom referenzierten Dateiformat, der Dateienanzahl und dem proprietären Konfigurationsmanagementsystem, was eine einfache Anpassung an verfügbare Konfigurationsmanagementwerkzeuge erlaubt. Eine Verwaltung oder die Versionskontrolle der XML-Datei wird nicht beschrieben.
Eine Dokumentation des Modells erfolgt mittels Textdokumenten, die in der XML-Datei zu einem Element bzw. Modellbestandteil referenziert werden. Die referenzierten Informatio-

nen werden über einen separaten Block im Modell innerhalb der Entwicklungsumgebung angezeigt und ermöglichen durch das Analysieren der XML-Datei die Erzeugung einer Komplettdokumentation für das Modell. In der Veröffentlichung werden keine Aussagen bzgl. des Versionsmanagements der Dateien oder der Abbildung von Modell- oder Parametervarianten getroffen. Ebenso ist unklar, wie mit der XML-Datei für die Modellkonfiguration umgegangen wird. Eine hierarchische Gliederung von XML-Dateien zur Wiederverwendung vorhandener und in sich abgeschlossener Container im Sinne eines übergeordneten Konfigurationsmanagements für mechatronische Fahrzeugsysteme mit den Komponenten Aktor, Sensor, Steuergerät und Streckenmodell, die in weitere Teilsysteme aufgeteilt sind, scheint nicht möglich zu sein. Das Arbeiten mit mehreren Entwicklern an einem Modell bedingt Aktualisierungen bzw. die Darstellung des Datei- bzw. Konfigurationsstatus der verteilten Elemente. Dies ist eine Grundvoraussetzung, um in Form von Softwarekoorperationen zusammen zu arbeiten, die jedoch nicht beschrieben wird.

## 2.3 Lokale Modelldatenbanken

In der Literatur ist mehrfach der Begriff „Modelldatenbank“ oder „Modellbibliothek“ zu finden. Dieser wird jedoch häufig im lokalen Kontext eines Entwicklers gebraucht, vgl. hierzu z. B. [Fu06], [Klo06], [Lin03], [Gu031]. Gemeint ist damit typischerweise die Ablage von Modellen auf dem lokalen Dateisystem des Entwicklers ohne Versions-, Varianten- und Konfigurationsmanagement in einer Bibliothek des Programmsystems. Oftmals erfolgt die Organisation der Simulationsmodelle in einer Bibliothek des Entwicklungswerkzeuges Matlab/Simulink. Die flexibel konfigurierbare Modellumgebung für die Fahrdynamiksimulation [Klo06] stellt beispielsweise eine komponentenbasierte Struktur zur Unterstützung in der Fahrdynamiksimulation zur Verfügung, wobei sich die unterschiedlichen Simulationsmodelle in der Modellierungstiefe unterscheiden. Eine Trennung zwischen Modell und Parameter ist realisiert und spezielle Blöcke zwischen den einzelnen Komponenten (sog. „Connector“) organisieren die Signalzusammenführung zwischen den nach außen geführten und internen Modellschnittstellen.

In den Veröffentlichungen [Gue03], [Gue031] und [Lin03] wird eine Modellverwaltung „VeLoDyn“ beschrieben, die ausschließlich für Matlab/Simulink eingesetzt wird. Zur Modellweitergabe wird ein Elementarblock als Träger für Simulink-Modelle bereitgestellt. Dieser bettet das Modell ein und stellt die Ein- und Ausgangssignale für die Umgebung bereit. Des Weiteren hängt dem Elementarblock[28] eine Datenbank an, die die notwendigen Informationen (Parameter, Version, Beschreibung und Supportfiles) für den eingebetteten Block bereithält. Unter einer Datenbank wird im diesem Kontext eine lokale für den Simulinkblock zugängliche Datenhaltung verstanden. Die „VeLoDyn“-Modelle finden derzeit ausschließlich in der Fahrzeuglängsdynamik zur Triebstrangsimulation und -applikation bei der IAV GmbH Anwendung. Mittels eines Signal- und Steuerflusses werden die Teil-

[28] Simulink-Block, der für die Informationsablage und den Informationstransport von „VeLoDyn“ implementiert und genutzt wird.

elemente miteinander gekoppelt. Grundlage für den Austausch von Modellen und Parametern ist eine Plattform der Modelle. Im Tool „VeloDyn“ wird eine pseudo-Standardisierung durch den Elementarblock erreicht.

## 2.4 Fazit

Die Betrachtung der Strukturierung und Entwicklung mechatronischer Fahrzeugsysteme (vgl. Abschnitt 2.1) macht deutlich, dass integrative Entwicklungsprozesse über mehrere Entwickler realisiert werden müssen. Der dafür erforderliche Modellaustausch und die einhergehende Transparenz zwischen den Entwicklern fordert eine zentrale Ablage der Modelle und der Parametersätze.
Eine Anwendung und Konfiguration der Teilelemente in einem übergeordneten System (Integrationsmodell) erfordert die Erweiterung der zentralen Ablage mit einer Versions-, Konfigurations- und Variantenkontrolle für die darin enthaltenen Elemente.
Eine Evaluierung durch die Literatur bekannter datenbankgestützter Entwicklungswerkzeuge hat gezeigt, dass bei Automobilherstellern und Zulieferern typischerweise hausinterne Eigenentwicklungen (z. B. bei BMW [Sch05] oder bei Continental Teves [Bun04]) zum Einsatz kommen. Weiterhin sind in unterschiedlichen Werkzeugen proprietäre Lösungen zur Ablage und Konfiguration von Modellen entstanden. Auch hier wird oftmals der Begriff (Modell-)Datenbank verwendet, der sich jedoch auf eine lokale dateibasierte Ablage bezieht. Die hausinternen Eigenentwicklungen verwenden in der Regel ein Versionskontrollsystem im Hintergrund und ermöglichen so eine Versionskontrolle der Elemente.
Eine genaue Analyse zeigt jedoch, dass diese Systeme auf die reine Funktionsentwicklung fokussieren. Die Integration der Streckenmodelle, die für die Funktionsabsicherung notwendig sind, werden oftmals nur am Rande berücksichtigt. Bezogen auf die Entwicklung mechatronischer Fahrzeugsysteme und die Wiederverwendung einmal getesteter Elemente wurden Defizite im Bereich der Konfigurations- und Variantenkontrolle festgestellt. Dies ist darin begründet, dass in den seltensten Fällen mit Dateigruppen gearbeitet wird und eine Abbildung von Referenzen zwischen den Elementen (Integrationsmodell ↔ Teilmodelle ↔ Parametervarianten) nicht vorhanden oder nur rudimentär ausgeprägt ist. Weiterhin ist die für integrative Entwicklungsprozesse so wichtige Transparenz (Kooperation und Kommunikation) beim Arbeiten in Softwarekooperation nur ansatzweise in einem der Systeme gegeben bzw. beschrieben.
Aus diesen Gründen sollen die Anforderungen für die Entwicklung mechatronischer Fahrzeugsysteme in Summe analysiert und identifiziert werden (vgl. Abschnitt 3.1). Diese stellen später die Basis für das Konzept und die Implementierung einer neuen Softwareumgebung zur speziellen Unterstützung der Entwickler mechatronischer Fahrzeugsysteme dar.

# 3 Konzeption der offenen Integrationsplattform „MeMo“ für mechatronische Fahrzeugsysteme

In den vorangegangenen Kapiteln wurde deutlich, dass eine Unterstützung für den Entwickler mechatronischer Fahrzeugsysteme und die damit verbundene Abbildung domänenübergreifender Entwicklungsprozesse maßgeblich durch ein zentrales Datenbanksystem erreicht werden kann. In Kapitel 2 erfolgte eine Evaluierung der bei unterschiedlichen Automobilherstellern und Zulieferern eingesetzten Systeme, soweit diese in der Literatur publiziert sind. In der Regel handelt es sich dabei um hausintern entwickelte Softwarewerkzeuge, die im Hintergrund ein Versionskontrollsystem verwenden. Durch die ergänzende Implementierung erfolgt eine benutzerspezifische Anpassung an die IT-Systeme am jeweiligen Einsatzort sowie eine Ausrichtung auf den Funktionsentwicklungsprozess. Die Evaluierung hat gezeigt, dass keines der betrachteten Werkzeuge auf die eingangs abgeleiteten Anforderungen (vgl. Abschnitt 2.1) zur Entwicklung mechatronischer Fahrzeugsysteme ausgerichtet ist (vgl. Abschnitt 2.4). In diesem Kapitel erfolgt daher eine Identifikation der Anforderungen zur Entwicklung mechatronischer Fahrzeugsysteme. Beginnend mit dem generischen Modell[29] und ausgeweitet auf das mechatronische Modell werden die Anforderungen identifiziert. Unter einem generischen Modell wird dabei ein Teilmodell eines mechatronischen Systems verstanden, das in sich geschlossen lauffähig ist. Das mechatronische Modell fasst typischerweise mehrere generische Modelle in einem übergeordneten Modell zusammen. Basierend auf den zusammengestellten Anforderungen erfolgt im Anschluss die Entwicklung eines Konzeptes für eine offene Integrationsplattform als datenbankbasierte Softwareumgebung zur Unterstützung der Entwickler mechatronischer Fahrzeugsysteme. Die Makroarchitektur des Konzeptes wird in Abschnitt 3.2 vorgestellt, bevor in Anschnitt 3.3 auf die resultierenden Anwendungsfälle[30] eingegangen wird. Die Anwendungsfälle stellen die Grundlage für eine Auswahl der Einzelkomponenten und eine spätere Implementierung sowie Bereitstellung dar.
Hauptaugenmerk dieses Kapitels ist es, die Anforderungen an eine Softwareumgebung zur Unterstützung der mechatronischen Modellbildung und der Funktionsabsicherung der Regelfunktion zu identifizieren. Hieraus erfolgen die Ableitung der Makroarchitektur für das Konzept und eine Beschreibung der Funktionalität auf Anwendungsfallebene für die resultierende neue Integrationsplattform „MeMo“.

[29] Als generisches wird ein Modell bezeichnet, wenn dieses durch Abstraktion gemeinsamer Merkmale und Eigenschaften in der Lage ist, in unterschiedlichem Kontext das gewünschte Verhalten zu erzeugen. Dieses kann durch diverse Datenvarianten ausgeprägt werden.

[30] Ein Anwendungsfall beschreibt anhand eines zusammenhängenden Arbeitsablaufes die Interaktion mit einem System. Ein Anwendungsfall wird stets durch einen Akteur initiiert und führt gewöhnlich zu einem für die Akteure wahrnehmbaren Ereignis [Oes05].

## 3.1 Anforderungen an eine offene Integrationsplattform

Basis für die Anforderungen an eine Integrationsplattform sind die Teilelemente (vgl. Abschnitt 3.1.1), aus denen das mechatronische System aufgebaut wird. Diese sind in der Regel Funktions- oder Streckenmodelle, die hierarchisch ineinander gegliedert sind (vgl. Abschnitt 2.1.2) und von einem oder mehreren Entwicklern bearbeitet werden können. Streckenmodelle stellen dabei vollständige CAE-Modelle zur virtuellen Abbildung von Bauteilen oder Zusammenhängen im Kraftfahrzeug dar (z. B. Fahrdynamik, Antriebsstrang). Als Funktionsmodell wird der Algorithmus im Steuergerät verstanden, der den elektronischen bzw. informationstechnischen Bestandteil mechatronischer Fahrzeugsysteme ausmacht. Neben der reinen Betrachtung der Modelle und Funktionen werden die Anforderungen durch die Arbeitsweise während der Entwicklung erweitert. Insbesondere die rechnergestützte domänenübergreifende Zusammenarbeit während der modellbasierten Entwicklung prägt mechatronische Systeme.

### 3.1.1 Generisches Modell

Ein generisches Modell bezeichnet das kleinste in sich abgeschlossene Modell, welches keine weiteren Teilmodelle hierarchisch gliedert. Ein Beispiel für ein generisches Modell stellt das Einspurmodell aus dem Bereich Fahrdynamik dar, das durch zwei Differentialgleichungen (vgl. Gleichung 3.1) beschrieben wird.

$$mv\dot{\beta} + (c'_{\alpha V} + c_{\alpha H})\beta + [mv^2 - (c_{\alpha H} l_H - c'_{\alpha V} l_V)]\frac{\dot{\psi}}{v} = c'_{\alpha V}\frac{\delta_L}{i} - F_{LY}$$

$$J_z\ddot{\psi} + (c'_{\alpha V} l_V^2 + c_{\alpha H} l_H^2)\frac{\dot{\psi}}{v} - (c_{\alpha H} l_H - c'_{\alpha V} l_V)\beta = c'_{\alpha V} l_V \frac{\delta_L}{i} - F_{LY} e_{SP}$$

**Gleichung 3.1 Lineares Einspurmodell mit konstanter Fahrgeschwindigkeit [Mit04]**

Ein Modell dieser Klasse wird typischerweise durch eine Vielzahl von Dateien beschrieben. Diese repräsentieren erst in Summe das Modell und sind deshalb als eine Einheit zu betrachten. Hierzu zählen unter anderem folgende Elemente:

- Modell
- Parameter
- Dokumentation
- Schnittstellenbeschreibung
- Ergebnisse der Validierung des Modells
- Zusätzlich benötigte Dateien, wie beispielsweise Bibliotheken

Das generische Modell kann als ein autark lauffähiges Modell oder Bestandteil eines mechatronischen Fahrzeugsystems bezeichnet werden. Folgende Anforderungen sind bei der kontextsensitiven Betrachtung durch MeMo zu erfüllen:

- Die Dateien in der Dateigruppe sowie die Dateigruppe als solche sind separat unter einer Versionskontrolle reproduzierbar zu verwalten.

- Die Ablage der Dateien und Dateigruppe muss für den orts- und zeitunabhängigen Austausch mit anderen Entwicklern bzw. Anwendern in einem zentralen Datenbanksystem erfolgen.
- Für die Weiterentwicklung muss eine steuerbare Frei- und Weitergabemöglichkeit für das Modell existieren.
- Der Status des Modells ist sowohl für den Anwender als auch für den Entwickler darzustellen. Dies kann entweder als Kennzeichnung an der Datei (mit Hilfe eines Symbols) oder direkt im Modell in der Entwicklungsumgebung erfolgen.
- Die Weiterentwicklung eines Modells muss durch die Statusdarstellung transparent für alle Anwender gemacht werden.
- Um eine zielgerichtete Anwendung zu einem späteren Zeitpunkt auch durch andere Entwickler zu gewährleisten, ist neben der zum Modell gehörenden Dokumentation eine kurze Beschreibung der wesentlichen Merkmale auf Dateigruppenebene erforderlich. Änderungen durch die Weiterentwicklung sind in Textform als Metadaten zu dem Modell zu speichern. Die Metadaten, die neben der Änderungshistorie auch Informationen zur Wiederauffindung, zum Modellverantwortlichen, zur Dokumentation (bzgl. der Auswahl im Datenbanksystem), zur Kennzeichnung von Release-Ständen und zum CAE-Werkzeug inklusive Version sowie dem Status enthalten, sind elementar für eine Organisation der Elemente in einem zentralen Datenbanksystem.
- Ein Modell kann durch die unterschiedliche Parametrisierung ein anderes Verhalten annehmen. Für die Gleichungen des Einspurmodells (vgl. Gleichung 3.1) kann durch Datenvarianten beispielsweise das Fahrdynamikverhalten eines Kleinwagens oder einer Luxuslimousine nachgebildet werden. Datenvarianten zu einem Modell sind separater Versionskontrolle zu unterstellen, da die Änderung bzw. Optimierung der Parameter keine Änderung am Modell bedeutet. Dies ist dadurch begründet, dass sich die Gleichung selbst nicht verändert und somit keine neue bzw. parallele Version des eigentlichen Modells (abgebildet durch Gleichungen) erzeugt werden muss. Eine Abbildung der Beziehung zwischen Modell und Parametersatz muss gewährleistet sein.
- Zusätzliche Datenvarianten müssen auf festgeschriebene Dateigruppen für Modelle abgebildet werden können.

### 3.1.2 Mechatronische Modelle zu Fahrzeugsystemen

Modelle für mechatronische Fahrzeugsysteme sind durch ihren interdisziplinären Charakter zusammengesetzte heterogene Modelle. Die Aufteilung der einzelnen Domänen des mechatronischen Systems erfolgt in der Regel auf einzelne Entwickler mit den entsprechenden Fachgebieten. Während der integrative Entwicklungscharakter dieser Systeme ein möglichst frühzeitiges Zusammenführen der Teilmodelle in einer Konfiguration oder dem sogenannten „Integrationsprojekt“ sowie das Zusammenarbeiten in Softwarekooperationen fordert, steht dem eine dezentrale Entwicklung in unterschiedlichen Organisationseinheiten des OEM oder beim Zulieferer gegenüber.

Das Integrationsmodell, welches die domänenspezifischen Teilelemente einbezieht dient dazu, die Teilelemente bzw. generischen Modelle in einer Konfiguration reproduzierbar zusammenzuhalten. Ebenso stellt dieses ein in sich geschlossenes Modell mit einer Vielzahl von Dateien und Verzeichnissen dar.
Die Liste der Anforderungen wird durch die Erweiterung der Betrachtung zusammengesetzter mechatronischer Modelle weiter ausgebaut:

- Ebenso wie das generische Modell ist das Integrationsmodell der Versionskontrolle zu unterstellen.
- Um eine getrennte Entwicklung der Teilelemente zu ermöglichen, sind diese unabhängig vom Integrationsmodell zu versionieren. Dies bedeutet eine Modularisierung des mechatronischen Gesamtmodells in der Art, dass eine separate Entwicklung der Teilelemente durch einzelne Entwickler stattfinden kann und der Bezug zwischen mechatronischem Gesamtmodell und den Teilmodellen erhalten bleibt. Eine mehrfache Datenhaltung für ein Teilelement ist aus Konsistenzgründen zu verhindern, was eine Referenzierung von Teilelementen in das Integrationsmodell mit Hilfe von bidirektionalen Verknüpfungen erfordert. Die referenzierten Teilelemente sind versionsgenau festzuhalten, um eine Reproduktion des Gesamtmodells gewährleisten zu können.
- Im Sinne der Zusammenarbeit mehrerer Entwickler an einem mechatronischen Fahrzeugmodell in Softwarekooperationen ist Transparenz über den aktuellen Entwicklungsstand aller Teilelemente erforderlich. Dies bedeutet, dass der Status der referenzierten Teilelemente im Integrationsprojekt ersichtlich ist und eine schrittweise Aktualisierung der einzelnen Teilelemente erfolgen kann, um einen Test weiterentwickelter Modelle durchführen zu können. Die Darstellung des Status der Teilelemente kann in der Entwicklungsumgebung des Anwenders oder direkt auf Dateiebene erfolgen.
- Die hierarchische Strukturierung mechatronischer Modelle (vgl. Abschnitt 2.1.2) muss eine übergeordnete Verwendung in einem neuen Integrationsmodell ermöglichen. Ebenso wie bei der Referenzierung von Teilelementen ist dieses in einem übergeordneten Integrationsmodell versionsgenau und reproduzierbar festzuhalten.
- Datenvarianten sind sowohl für mechatronische Fahrzeugsysteme als auch für einzelne generische Modelle möglich. Für die mechatronischen Fahrzeugsysteme als übergeordnete Instanz eines generischen Modells oder anderer mechatronischer Teilelemente kann in dem separat versionierten Parametersatz eine Referenz zu einem existenten Parametersatz eines generischen Modells oder eines mechatronischen Fahrzeugsystems existieren.

### 3.1.3 Zentrale Ablage für das modellbasierte Entwickeln

Um einen modellbasierten Entwicklungsprozess mechatronischer Fahrzeugsysteme über verschiedene organisatorische Einheiten hinweg zu ermöglichen, ist die zentrale Ablage der Modelle in einem Datenbanksystem unumgänglich [Bei97], [Fis83], [Enc90]. Über die Stan-

dardmerkmale eines Versionskontrollsystems hinaus sind weiterhin Möglichkeiten für ein Konfigurations- und Variantenmanagement von CAE-Modellen gefordert. Die Besonderheiten im Umgang von CAE-Modellen innerhalb mechatronischer Systeme wurden in den Abschnitten 3.1.1 und 3.1.2 für generische und mechatronische Modelle beschrieben. Neben diesen sind für die modellbasierte Entwicklung des Datenbanksystems folgende zusätzliche Anforderungen zu erfüllen:

- Metadaten zur textuellen Dokumentation der Dateien und Dateigruppen müssen frei und passend für die Organisation von Funktions- und Simulationsmodellen wählbar sein.
- Persistentes[31] Speichern der Metadaten auf einzelne Dateien und Dateigruppen.
- Zur Strukturierung der Elemente ist dem Anwender eine Struktur, die sich anhand mechatronischer Merkmale orientiert, bereitzustellen. Diese ermöglicht einfaches und effizientes wieder Auffinden vorhandener Elemente.
- Um eine Nutzer- und Rechteverwaltung einfach zu handhaben ist eine Integration in die IT-Struktur des Unternehmens notwendig.
- Bereitstellung einer plattformunabhängigen Schnittstelle für den Zugriff auf das Datenbanksystem und damit Möglichkeit zur Anbindung an die CAE-Werkzeuge des Anwenders.
- Die Entwicklung von Funktionssoftware als Teil einer Steuergerätefunktion wird basierend auf den Anforderungen in einem Anforderungsmanagementsystem durchgeführt. Daher ist es besonders wichtig, eine softwaretechnische Verknüpfung zwischen dem Funktionsmodell einerseits und der Spezifikation andererseits herzustellen. Der Test der entwickelten Funktionssoftware erfolgt im Idealfall möglichst früh im Entwicklungsprozess durch die Simulation der Regelfunktion im Verbund mit einem Streckenmodell. Das dazu verwendete Streckenmodell sowie die Testfälle sind einer Versionskontrolle zu unterziehen. Testfälle sind möglichst generisch aufzubauen und sowohl mit der Funktionssoftware als auch mit dem Streckenmodell softwaretechnisch zu verknüpfen. Zum Fokus der vorliegenden Arbeit zählt insbesondere die Unterstützung des Modellbildungsprozesses. Dennoch ist für das Konzept eine zukünftige softwaretechnische Anbindung an ein IT-System zum Anforderungs- und Testfallmanagement zu berücksichtigen.
- Arbeiten mit lokal vorhandenen Daten ohne eine aktive Verbindung zum Datenbanksystem.

### 3.1.4 CAE-Werkzeug zur rechnergestützten Modellbildung und Simulation

Die Entwicklung und Applikation mechatronischer Fahrzeugsysteme erfolgt unter Anwendung rechnergestützter Methoden in diversen CAE-Werkzeugen. Diese werden typischerweise abhängig von der Domäne durch den Entwickler selbst gewählt. Für die Reglerent-

[31] Dauerhafte Speicherung der Daten oder -strukturen in nicht-flüchtigen Speichermedien wie z. B. einem Datenbanksystem.

wicklung und die Modellbildung von Streckenmodellen wird von vielen das Werkzeug Matlab/Simulink verwendet[32]. Der Entwickler soll in den von ihm verwendeten Werkzeugen unterstützt werden. Dabei lassen sich die Anforderungen bzgl. der CAE-Werkzeuge in zwei Kategorien aufteilen. Die Anbindung eines zentralen Datenbanksystems erfordert softwaretechnische Rahmenbedingungen, die unter dem Punkt „Softwaretechnische Schnittstelle“ zusammengefasst sind. Zusätzlich bestehen Anforderungen aus Sicht des Entwicklers, die den integrativen Entwicklungsprozess fokussieren. Zusammengefasst sind diese im Punkt „Interaktion mit dem Datenbanksystem“.

**Softwaretechnische Schnittstelle**

- Möglichkeit zur Anbindung des zentralen Datenbanksystems an das CAE-Werkzeug.
- Bidirektionale Kommunikation zwischen dem CAE-Werkzeug und dem zentralen Datenbanksystem.
- Skriptbasierte Kommunikation mit dem Datenbanksystem zur Implementierung von Algorithmen. Besteht die Option einer skriptbasierten Interaktion mit dem CAE-Werkzeug sind automatisierte Testszenarien zukünftig denkbar.

**Interaktion mit dem Datenbanksystem**

- Zugriff auf die im zentralen Datenbanksystem vorhandenen Elemente.
- Anwendung der im Datenbanksystem vorhandenen Elemente (Simulation, Parametrisierung, Konfiguration etc.).
- Steuerung der Mechanismen zur Versionierung, Konfiguration von Systemen und Ablage von Datenvarianten.
- Darstellung des Status verwendeter Elemente (z. B. „Aktuelle Version“ oder „Neue Version vorhanden“).
- Schrittweise Aktualisierung der Teilelemente in einem Integrationsmodell.

Ein CAE-Werkzeug als Bestandteil im Konzept der offenen Integrationsplattform MeMo (vgl. Abschnitt 3.2) zur Unterstützung des mechatronischen Entwurfs wird in Abschnitt 4.2 evaluiert.

## 3.2 Makroarchitektur des Konzeptes

Wie in Kapitel 2 festgestellt, erfordert die Implementierung die Integration eines Datenbanksystems in das CAE-Werkzeug, in dem der Anwender unterstützt werden soll. Dieses ist wiederum von der Modellierungsdomäne abhängig. Das Entwicklungswerkzeug Matlab/Simulink wird im Kontext der Modellbildung und Funktionsentwicklung sehr häufig angewendet (vgl. Abschnitt 4.2). Dennoch sind andere Entwicklungswerkzeuge wie z. B.

[32] Als Entwicklungsumgebung zum Aufbau und zur Verwendung von systemdynamischen Simulationsmodellen hat sich die blockschaltbildorientierte Simulationsplattform MATLAB/Simulink in den letzten Jahren auf breiter Front durchgesetzt und dabei quasi zum Industriestandard entwickelt [Ise06].

ADAMS oder Modelica/Dymola zu berücksichtigen. Das Konzept sowie eine spätere Implementierung werden vorerst mit dem Entwicklungswerkzeug Matlab/Simulink forciert und damit exemplarisch umgesetzt. Zu einem späteren Zeitpunkt erfolgt die Integration in andere CAE-Werkzeuge. Um dies gewährleisten zu können, ist eine von dem CAE-Werkzeug unabhängige Technologie mit einer Kapselung[33] der speziellen Funktionen zur Unterstützung der Modellbildung mechatronischer Fahrzeugsysteme notwendig, die unabhängig vom dem Werkzeug funktionieren kann. Im optimalen Fall sind diese derart gekapselt, dass eine Integration der Funktionalität in diverse CAE-Werkzeuge möglich ist. Dadurch kann für den Anwender der maximale Komfort erreicht werden, wenn dieser Interaktionen direkt aus dem ihm bekannten Werkzeug heraus veranlassen respektive steuern kann. Eine denkbare Lösung ist die Einbindung des Datenbanksystems und der zusätzlichen Funktionalität für mechatronische Fahrzeugsysteme mit Hilfe einer Middleware[34] in die CAE-Werkzeuge der Anwender. Die Middleware eines plattformunabhängigen Konzeptes basiert auf der Programmiersprache Java[35] und erfüllt folgende Aufgaben:

- Entkopplung des Datenbanksystems von dem Entwicklungswerkzeug und damit Berücksichtigung der unterschiedlichen Schnittstellen des Datenbanksystems und der CAE-Werkzeuge.
- Bildung einer Fassade[36] zur Integration der zusätzlichen Funktionen, die zugeschnitten sind auf die Entwicklung mechatronischer Fahrzeugsysteme in unterschiedlichen CAE-Werkzeugen.
- Autarke Kapselung der Funktionalität zur eigenständigen Anwendung oder Einbindung in andere im Prozess beteiligte Softwarewerkzeuge.
- Bestmögliche Aufteilung der Algorithmen/Funktionalitäten in unterschiedliche Komponenten wie Datenbanksystem, Middleware und CAE-Werkzeug.
- Implementierung der Funktionalität in einem vom Entwicklungswerkzeug unabhängigen Programmcode.

Die nachstehende Abbildung 3.1 stellt das gewählte Konzept mit dem Datenbanksystem, der Middleware „MeMo-Plattform“ und einem exemplarischen Entwicklungswerkzeug dar. Die MeMo-Plattform als Kernbaustein des Konzeptes erfüllt die oben summierten Anforderungen und verbindet einzigartig ein Datenbanksystem mit diversen Entwicklungswerkzeugen. Dies ermöglicht den Durchgriff des Anwenders aus dem Entwicklungswerkzeug auf zentral abgelegte Modelle, Parameter und Metainformationen.

---

[33] Kapselung ist ein Paradigma der objektorientierten Programmierung. Als Kapselung bezeichnet man eine definierte, nach außen geöffnete Schnittstelle, die einen kontrollierten Zugriff auf Methoden bzw. Attribute von Klassen gewährleistet.

[34] Eine Middleware bezeichnet in der Informatik anwendungsunabhängige Technologien, die Dienstleistungen zur Vermittlung zwischen Anwendungen anbieten, so dass die Komplexität der zugrundeliegenden Applikation und Infrastruktur verborgen bleibt [Ruh00].

[35] Objektorientierte, in starkem Maße an C++ angelehnte betriebsystem- und plattformunabhängige Programmiersprache, die 1995 von der Firma Sun Microsystems (USA) entwickelt wurde [VDI2206], [Bal00].

[36] Eine Fassade ist ein Entwurfsmuster aus dem Bereich der Softwareentwicklung und gehört zu der Kategorie der Strukturmuster. Es bietet eine einheitliche und vereinfachte Schnittstelle zu einer Menge von Schnittstellen eines Subsystems. Die Fassade definiert eine Schnittstelle auf höherer Ebene und ermöglicht die einfache Nutzung der darin bereitgestellten Subsysteme [Gam95].

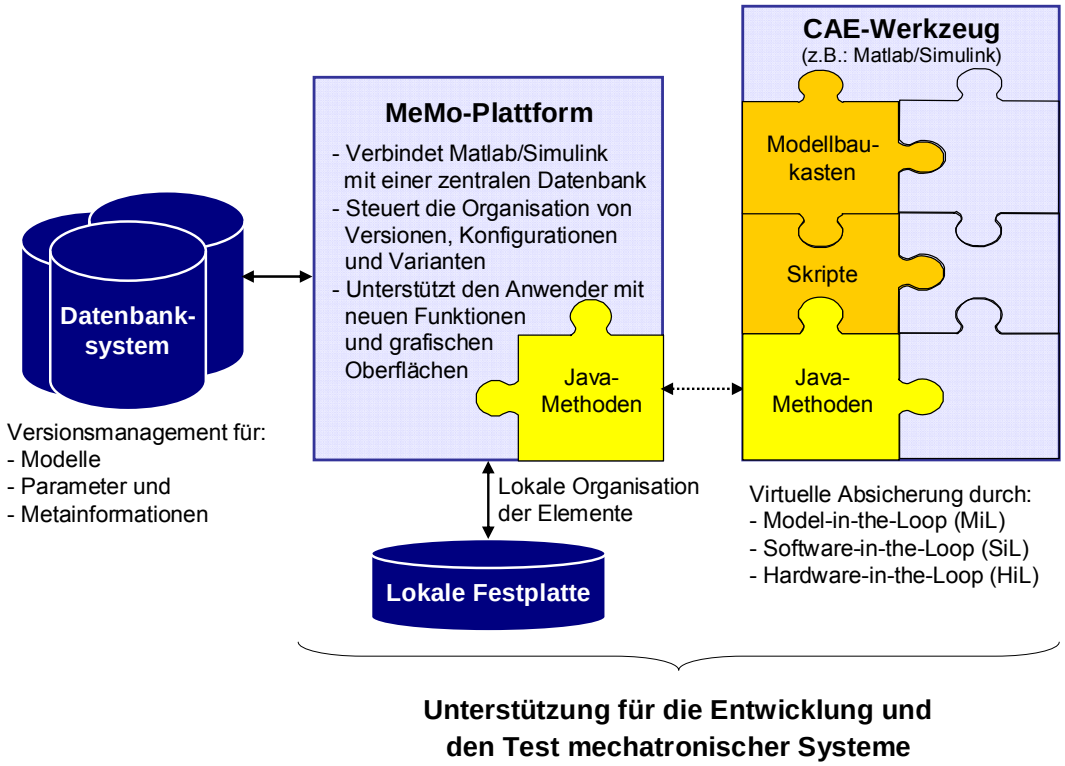


**Abbildung 3.1 - Konzept der offenen Integrationsplattform "MeMo"**

Die Auswahl eines geeigneten Datenbanksystems für das Versionsmanagement von Modellen, Parametern und Metadaten mechatronischer Modelle erfolgt in Kapitel 4.1. Das ausgewählte Datenbanksystem, das mindestens ein Versionsmanagement bereitstellen muss, ist von der MeMo-Plattform zu steuern. Implementiert sind in der MeMo-Plattform weiterhin alle Algorithmen für eine Organisation von Konfigurationen (im Modellierungskontext: Integrationsprojekte) und optionalen Datenvarianten eines Modells. Ferner ist darin die Abbildung des für mechatronische Systeme typischen integrativen Entwicklungsprozesses umgesetzt. Neben der Kommunikation mit dem Datenbanksystem übernimmt die MeMo-Plattform des Weiteren die Kommunikation mit dem CAE-Werkzeug (z. B. Matlab/Simulink). Die bidirektionale Kommunikation ermöglicht dem Entwickler aus dem CAE-Werkzeug heraus die Anforderung von Dateien bzw. den Aufruf von verfügbaren Anwendungsfällen durchzuführen. Die einzelnen Elemente, das Datenbanksystem und das CAE-Werkzeug werden durch die MeMo-Plattform miteinander verbunden. Die MeMo-Plattform ist die Kern-Komponente der gesamten Architektur und stellt als Middleware den zentralen Baustein für die notwendige Kommunikation zwischen Datenbanksystem, lokaler Festplatte des Klienten und dem CAE-Werkzeug her. Die darin enthaltenen Algorithmen steuern die Organisation mechatronischer Teilmodelle und ermöglichen so ein Management von Versionen, Varianten und Konfigurationen. Ein Aufruf der darin enthaltenen Funktionalität kann durch den Anwender wahlweise über die grafischen Oberflächen der MeMo-Plattform oder über Funktionsaufrufe aus dem Entwicklungswerkzeug selbst veranlasst werden. Ein Aufruf aus dem CAE-Werkzeug bedient die Fassadenklasse der

MeMo-Plattform und stellt für den Anwender ähnliche grafische Oberflächen wie ein direkter Aufruf innerhalb der MeMo-Plattform bereit. Eine Zusammenstellung der in der MeMo-Plattform benötigten Funktionalitäten erfolgt in Anwendungsfall-Diagrammen. Diese werden im nachstehenden Abschnitt 3.3 abgehandelt.

## 3.3 Anwendungsfälle der offenen Integrationsplattform

Basierend auf den identifizierten Anforderungen (vgl. Abschnitt 3.1) und der im Entwicklungsprozess notwendigen Aktivitäten mit Modellen werden Anwendungsfälle zur Spezifikation des Funktionsumfangs erarbeitet. Diese stellen den Ausgangspunkt für die Implementierung der neuen Software dar. Bezogen auf die Nutzer, denen die Funktionalität aus den Anwendungsfällen bereitgestellt wird, lässt sich aufgrund der Tätigkeit eine Klassifizierung in verschiedene Rollen vornehmen:

- Anwender: Mitglieder dieser Gruppe wenden in dem Datenbanksystem vorhandene Modelle für ihren Aufgabenbereich an. Typischerweise werden Parameter angepasst, um ein existierendes Modell in einem bestimmten anwendungsspezifischen Kontext zu verwenden.
- Entwickler: Die Gruppe der Entwickler modelliert Funktionsmodelle für Steuergeräte und Streckenmodelle zum Test dieser. In der Regel ist ein Entwickler für ein oder mehrere Teilmodelle oder ein mechatronisches Fahrzeugsystem verantwortlich.
- Projektleiter: Der Projektleiter koordiniert bzw. administriert Entwicklungsprojekte, an denen mehrere Entwickler beteiligt sind.
- Administrator: Verwaltet die Elemente im Datenbanksystem und ist für die Zuordnung der einzelnen Nutzer in die jeweiligen Nutzergruppen zuständig.

Eine Anmeldung am Datenbanksystem ermöglicht eine Zuordnung der Nutzer auf eine oder mehrere Rollen. Abhängig von der Berechtigung auf Elemente in dem Datenbanksystem, die auf Rollen-Ebene vergeben werden, stehen dem authentifizierten Nutzer für das Element die entsprechenden Handlungsoptionen zur Verfügung. Für die oben beschriebenen Rollen werden Anwendungsfälle entworfen, die die Interaktion des einzelnen Nutzers mit der offenen Integrationsplattform beschreiben. Hierbei wird die Gesamtfunktionalität der neuen Softwareumgebung überblicksartig aufgezeigt. Die neuen Anwendungsfälle, die speziell für die Entwicklung mechatronischer Fahrzeugsysteme konstruiert werden, sind im Detail erörtert.

### 3.3.1 Anwendungsfälle: Anwender

Der geringste Funktionsumfang ist für den Nutzer in der Rolle „Anwender“ vorgesehen. Den Nutzern in den Rollen der Entwickler, Projektleiter und Administratoren wird der Funktionsumfang hierarchisch vererbt. Gleichzeitig stellt der mit der Rolle der „Anwender“ verbundene Funktionsumfang, der in Abbildung 3.2 dargestellt ist, die Basis für das Arbei-

ten aller Nutzergruppen mit der neuen Integrationsplattform in Form einer datenbankbasierten Softwareumgebung dar.

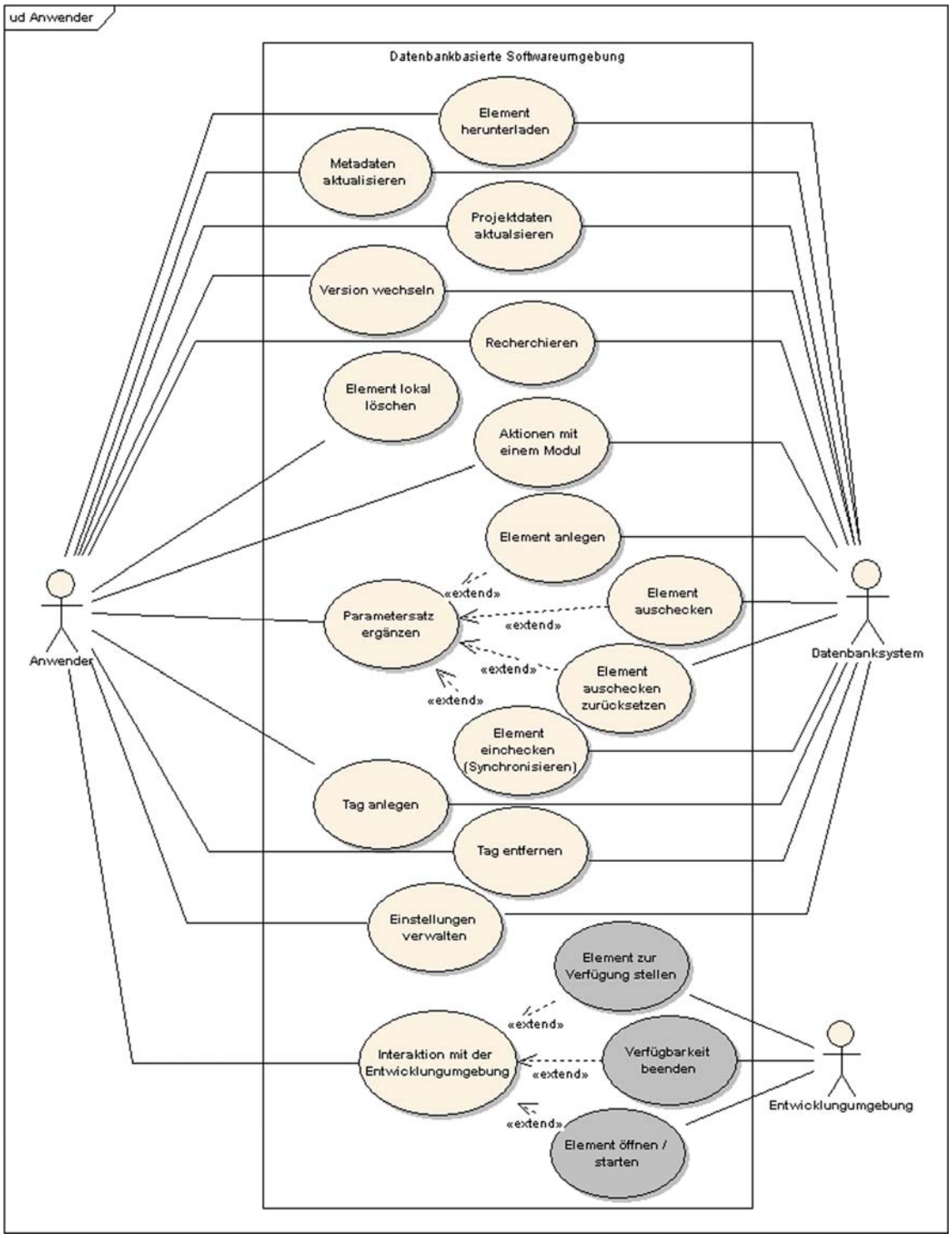


**Abbildung 3.2 - Anwendungsfalldiagramm mit der Nutzerrolle „Anwender“**

Der Funktionsumfang ist so gewählt, dass die im Datenbanksystem existenten Elemente in dem Entwicklungswerkzeug zur Anwendung gebracht werden können. Hierin inbegriffen ist die Auswahl des richtigen Elements durch entsprechende Strukturierung des Datenbe-

standes, einer Suchfunktion und der Dokumentation in Form von Metadaten zu jeder verfügbaren Dateigruppe. Wie bereits in den Anforderungen (vgl. Abschnitt 3.1.1) beschrieben, erfolgt eine Anpassung des Modellverhaltens in der Simulation typischerweise durch eine Variation der Parameter. Diese Option wird in der Rolle des „Anwenders“ benötigt, um ein anwendungsspezifisches Verhalten in dem Datenbanksystem vorhandener Modelle zu ermöglichen. Der daraus resultierende, neu erstellte Parametersatz für das Simulationsmodell muss vom Anwender persistent unter Versionskontrolle gespeichert werden können. Darüber hinaus ist in der Datenbank eine versionsgenaue Referenz zwischen Modell und Parametersatz aufzubauen. Dies ermöglicht anderen Anwendern des Modells den Parametersatz selbst oder in einem übergeordneten Modell zu verwenden. Die Anwendung eines Modells (ggf. inklusive eines Parametersatzes) erfolgt in einem CAE-Werkzeug (z. B.: Matlab/Simulink). Typischerweise muss diesem der lokale Pfad zu den einzelnen Dateien bekannt sein, um eine Simulation durchzuführen, was entsprechende Interaktionen zwischen der MeMo-Plattform und dem CAE-Werkzeug erfordert. Nach erfolgreicher Simulation werden Mechanismen zur Dokumentation der verwendeten Modelle und Parametersätze benötigt. Hier ist darauf zu achten, dass dies auch für Modelle bzw. Elemente erfolgen muss, auf die der Anwender kein Recht zur Weiterentwicklung besitzt. Deshalb ist eine Dokumentation mit Hilfe von sog. „Tags“ [37] zu implementieren. Durch Tags ist es möglich, eine beliebige – auch festgeschriebene – Version mit einem Hinweis (Nutzername, Status, Datum, Kommentar) auf Metadatenebene zu kennzeichnen. Das stellt für das zukünftige wieder Auffinden des Elements einen entscheidenden Faktor dar.

Im Konzept von MeMo (vgl. Abbildung 3.1) wird die Anbindung der lokalen Festplatte an die MeMo-Plattform gezeigt, auf der die lokal vom Anwender angewendeten Modelle gespeichert werden. Die lokal vorhandenen Elemente sind über die MeMo-Plattform zu verwalten. Hier ist vor dem Hintergrund interaktiver Entwicklungsprozesse mechatronischer Fahrzeugsysteme die Statusdarstellung für jedes Element besonders wichtig. Über den Status eines Elementes ist für den Anwender zu erkennen, ob Teilelemente in einer neueren Version im Datenbanksystem vorliegen. Für die Transparenz zwischen den an einem mechatronischen Fahrzeugmodell beteiligten Entwicklern bzw. Modellanwendern ist dies ein wesentliches Merkmal, um die Aktualität der Teilelemente sicherzustellen. Basierend auf der Änderungsbeschreibung von einer Version zur nächsten kann der Anwender über die Verwendung in seinem Kontext entscheiden und bei Bedarf die lokal vorhandene Version mit der aktuellen Version aus dem Datenbanksystem austauschen.

Zusammenfassend wurden 19 Anwendungsfälle für die Rolle der „Anwender“ identifiziert. Diese lassen sich in fünf Kategorien gliedern:

- Element anwenden: „Element herunterladen“, „Version wechseln“, „Elemente in Matlab/Simulink verfügbar machen“, „Element in Matlab öffnen“, „Verfügbarkeit in Matlab/Simulink beenden“ und „Metadaten oder Projektdaten aktualisieren“
- Parametersatz anlegen: „Parametersatzobjekt anlegen“ und „Parametersatzkonfiguration anlegen“

[37] In der Datenverarbeitung und Informatik steht das englische Wort „Tag“ (deutsch: Etikett, Anhänger, Aufkleber, Marke, Auszeichner) für die Kennzeichnung eines Datenbestandes mit zusätzlichen Informationen [Wik01].

- Mit einem Modul arbeiten: „Modul erstellen, „Modul bearbeiten“ und „Modul löschen“
- Dokumentation: „Tag erstellen“ und „Tag löschen“
- Allgemein: „Recherchieren“ und „Einstellungen verwalten“

### 3.3.2 Anwendungsfälle: Entwickler

Der Entwickler, dessen Aufgabe durch die Modellbildung physikalischer Systeme und Modellierung von Steuergerätefunktionen definiert ist, benötigt Funktionalitäten zur Erstellung neuer Elemente im Datenbanksystem. Über das Funktionsspektrum aus der Rolle des Anwenders hinaus wird für den Nutzer als Mitglied in der Gruppe „Entwickler“ das Funktionsspektrum um die in Abbildung 3.3 dargestellten Anwendungsfälle erweitert.

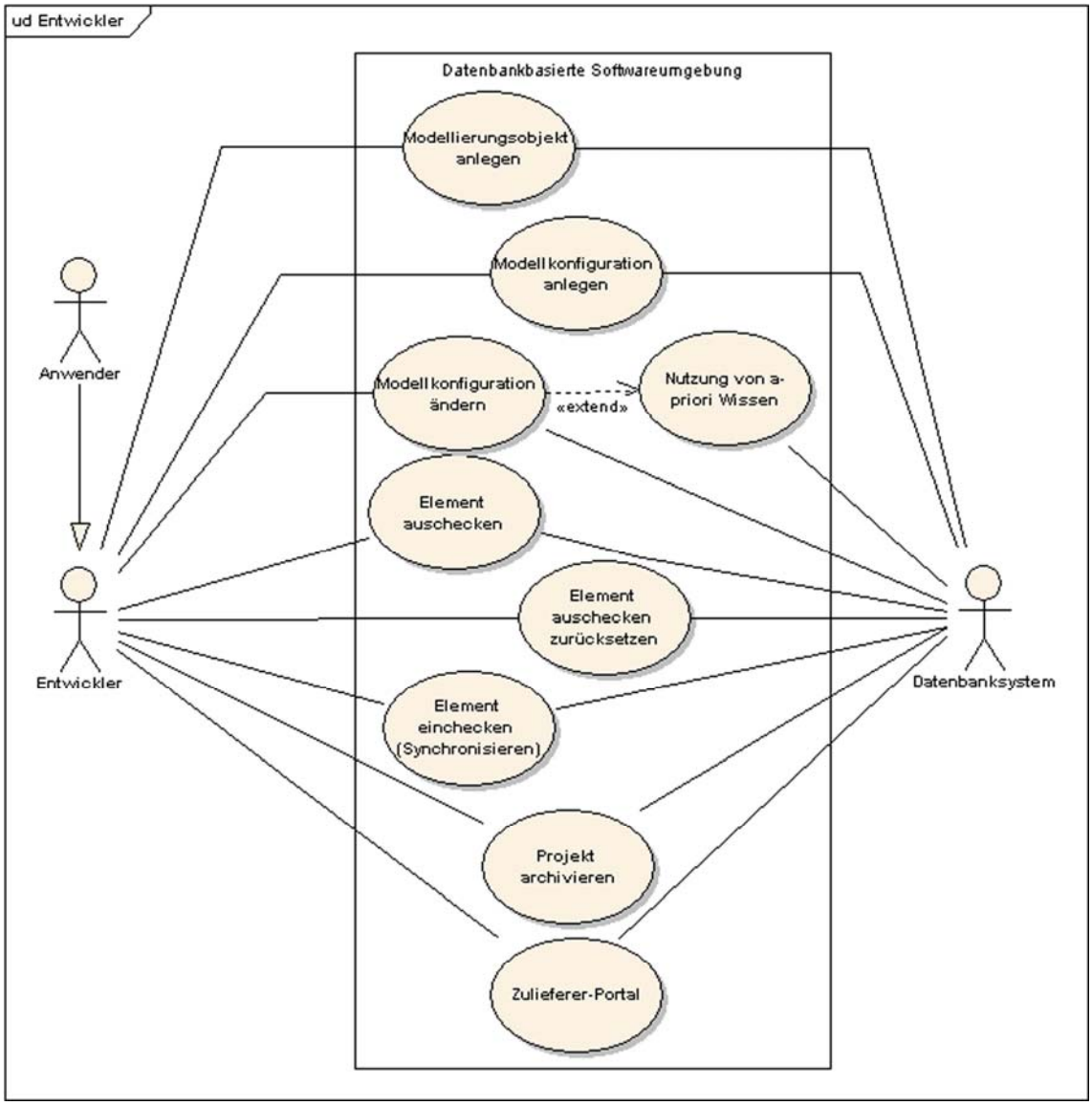


**Abbildung 3.3 - Anwendungsfalldiagramm mit der Nutzerrolle „Entwickler“**

Besonderes hervorzuheben sind hier die Anwendungsfälle „Modellierungsobjekt oder Modellkonfiguration anlegen“, „Modellkonfiguration ändern“ und Nutzung von a-priori-Wissen. Diese unterstützen den Aufbau von hierarchisch gegliederten Fahrzeugsystemen (Integrationsprojekte). Teilelemente werden dabei als Modellierungsobjekte und Konfigurationen als Modellkonfigurationen bezeichnet. Diese sind separat versionierbar und können innerhalb einer Modellkonfiguration verlinkt werden. Dabei darf keine Kopie des Elements erstellt werden. Die verlinkten Elemente sind versionsgenau festgehalten, was eine Reproduktion eines Fahrzeugsystems jederzeit mit den gleichen Teilelementen erlaubt.

Zum Aufbau von Modellkonfigurationen kann auf bereits vorhandenes Wissen im Datenbanksystem zurückgegriffen werden (Anwendungsfall „Nutzung von a-priori-Wissen“). Die Herausforderung für den Entwickler liegt darin, bereits vorhandene Elemente in dem Datenbanksystem als mögliche Bestandteile für eine neue Modellkonfiguration zu identifizieren bzw. zielgerichtet zu definieren. Hier müssen insbesondere die Schnittstellen und die darüber übertragenen Informationen (Einheit, Skalierung etc.) harmonieren. Um den Entwickler diesbezüglich zu unterstützen, kann auf die in der Vergangenheit aufgebauten Modellkonfigurationen zurückgegriffen werden. Unter der Annahme, dass die darin verwendeten Teilelemente bezüglich ihrer Signalschnittstellen zusammen verwendet werden können sind Empfehlungen für den Entwickler bereitzustellen. Sobald ein einzelnes Teilelement in die Modellkonfiguration verlinkt ist, kann nach den Elementen recherchiert werden, die bereits zuvor in anderen Fahrzeugsystemen miteinander verbunden waren. Für die nach der Recherche angebotenen Teilelemente kann nicht gewährleistet werden, dass die darin implementierten Schnittstellen zueinander passen, daher ist das Ergebnis als „Empfehlung“ zu betrachten.

Da die Funktionsentwicklung von Steuergerätefunktionen oftmals von externen Partnern übernommen wird, sind Funktionen innerhalb der MeMo-Plattform gefordert, die einen Export eines Projektes erlauben. Nach der externen Entwicklung und bei der Lieferung von Zwischenständen ist ein automatischer Import der gelieferten Dateien auf dasselbe Projekt durchzuführen. Idealerweise wird dem externen Partner hierzu ein Webportal[38] zur Verfügung gestellt, mit dem er Ergebnisse selbstständig bereitstellen kann. Im Falle einer Lieferung ist der Entwickler, der den Export durchführte, per E-Mail zu benachrichtigen. Das Vorgehen stellt sicher, dass von externen Partnern gelieferte Steuergerätefunktionen in einer frühen Phase des Entwicklungsprozesses für die Entwickler und Anwender zur Verfügung stehen. Abhängig von den Berechtigungen auf das Projekt werden somit Funktionstest mittels MiL und SiL in unterschiedlichen Domänen ermöglicht.

[38] Ein Webportal bündelt regelmäßig benötigte Dienste zu einem Themenkomplex. Die Bereitstellung erfolgt über eine Internetseite, von der aus die entsprechenden Dienste gestartet werden. Im vorliegenden Fall stellt das Webportal eine Schnittstelle mit Entkopplungscharakter zwischen den Daten im Unternehmen und extern (im Internet verfügbaren Diensten) dar.

### 3.3.3 Anwendungsfälle: Projektleiter

In der Benutzerrolle des „Projektleiters“ stehen neben den Anwendungsfällen der „Entwickler“ noch weiterführende Funktionalitäten zur Verfügung. Die Mitglieder in der Gruppe „Projektleiter“ sind ebenfalls in einer Gruppe der Entwickler enthalten. Die für den Projektleiter identifizierten Anwendungsfälle unterstützen die Leitung von Modellierungsprojekten, an dem mehrere Entwickler parallel zusammenarbeiten. Das durch die zur Verfügung gestellten Anwendungsfälle zu erreichende Ziel ist die Koordination der Entwicklung eines mechatronischen Fahrzeugprojektes. Abbildung 3.4 fasst die einzelnen Anwendungsfälle in einem Anwendungsfalldiagramm zusammen.

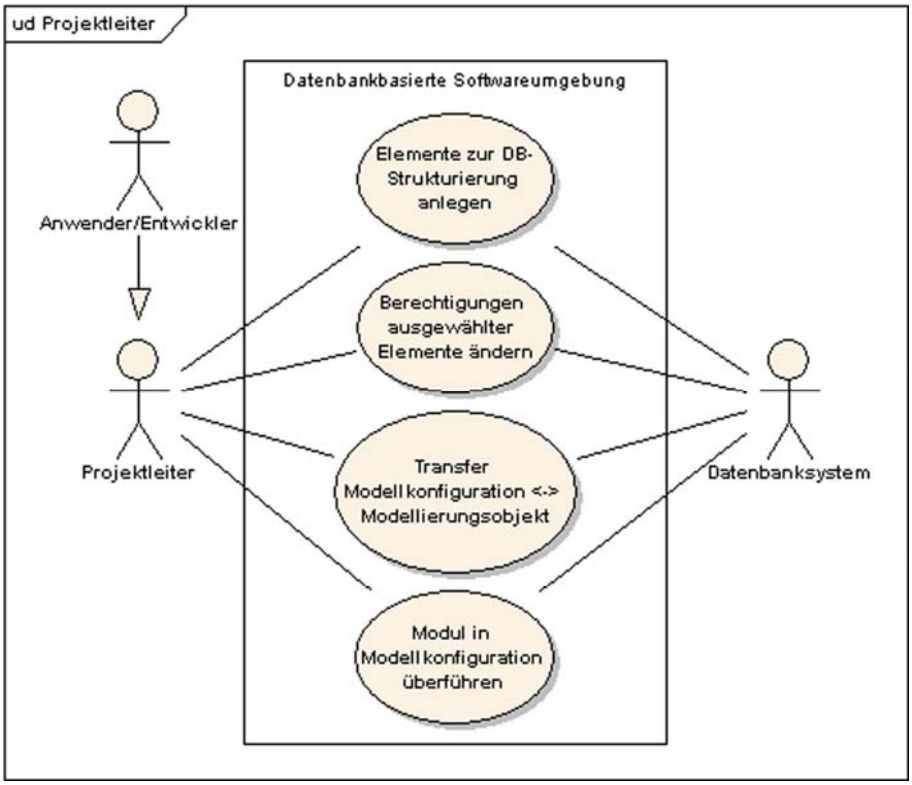


**Abbildung 3.4 - Anwendungsfälle mit der Nutzerrolle „Projektleiter“**

Durch die dem Projektleiter zur Verfügung stehenden Anwendungsfälle der Gruppe „Anwender/Entwickler“ kann durch diese Gruppe ein neues mechatronisches Fahrzeugsystem aufgesetzt werden, was die Erzeugung der einzelnen Modellierungsobjekte und der Modellkonfigurationen sowie ein initiales Setzen der Berechtigungen (Anwendungsfall „Berechtigungen ausgewählter Elemente ändern“) beinhaltet. Fehlende Datenbankzweige zum Speichern der Elemente können über den Anwendungsfall „Elemente zur Datenbankstrukturierung anlegen“ ergänzt werden. Die erzeugten Teilelemente können wahlweise in einem Modul oder einer Modellkonfiguration für die tägliche Arbeit der beteiligten Entwick-

ler zusammengeführt werden. Um ein Release[39], basierend auf einem Modul, ohne den Versionsbezug der darin enthaltenen Teilelemente zu erzeugen, kann das Modul in eine Modellkonfiguration überführt werden (Anwendungsfall „Modul in Modellkonfiguration überführen“). Die Modellkonfiguration hält die darin enthaltenen Teilelemente mit einem expliziten Versionsbezug fest und ermöglicht so eine Reproduktion dieser zu jedem Zeitpunkt. Während der Entwicklungsphase eines mechatronischen Fahrzeugsystems wird eine Anpassung der hierarchischen Struktur des Gesamtmodells durch den Anwendungsfall „Transfer Modellkonfiguration ↔ Modellierungsobjekt“ unterstützt.

### 3.3.4 Anwendungsfälle: Administrator

Im Gegensatz zu den Aufgaben des Projektleiters sind von der Nutzergruppe „Administrator“ administrative Aufgaben durchzuführen. Hierzu zählen auf Modell-Ebene eine Konsistenzprüfung aller in der Datenbank verfügbaren Elemente, aber auch die Verwaltung der Berechtigungen und die Zuordnung der einzelnen Nutzer in die entsprechenden Gruppen. Die Abbildung 3.5 zeigt das zugehörige Anwendungsfalldiagramm der MeMo-Plattform.

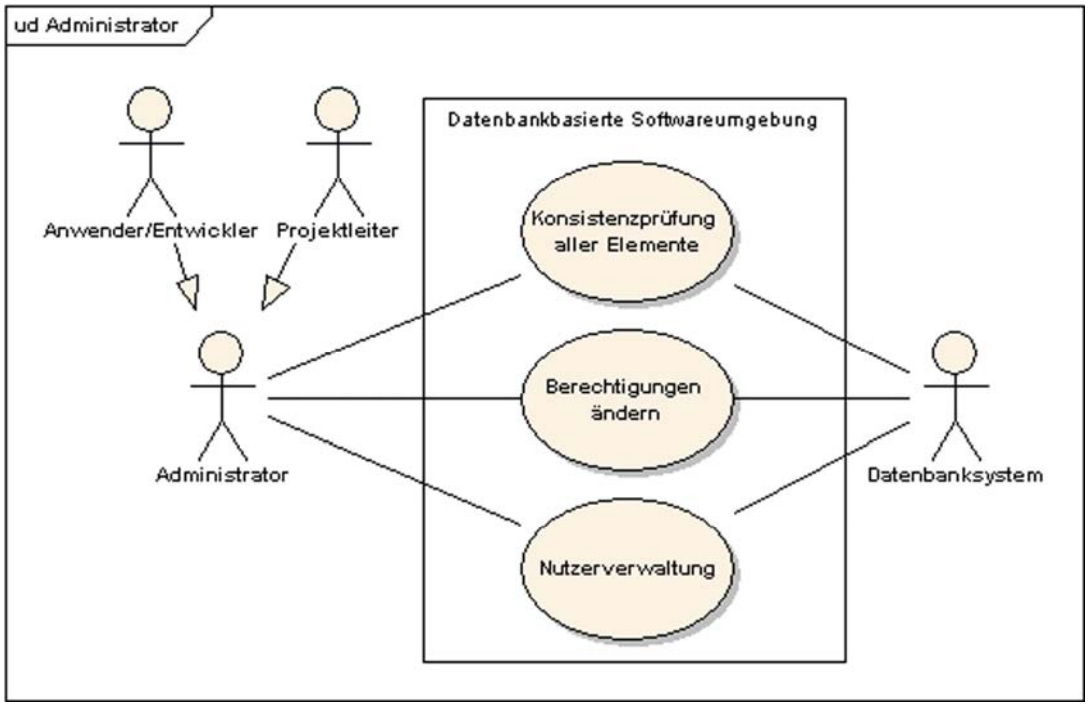


**Abbildung 3.5 - Anwendungsfälle mit der Nutzerrolle „Administrator“**

[39] Der Begriff Release stammt aus der Softwaretechnik und bezeichnet einen fertigen und veröffentlichten Versionsstand einer Software. Im Kontext der modellbasierten Entwicklung wird unter Software ein Strecken- oder Funktionsmodell verstanden.

# 4 Festlegung der Komponenten für das Konzept von „MeMo“

Das in Kapitel 3 vorgestellte Makrokonzept mit der MeMo-Plattform als Kernbaustein wird von einem Datenbanksystem für das Versionsmanagement von Dateien, Projekten (Modelle und Parameter) und Metadaten unterstützt. Die Integration in die Entwicklungswerkzeuge erfolgt auf Softwareebene und soll nach der Implementierung eine interaktive Arbeitsweise des Anwenders in seinem gewohnten Entwicklungswerkzeug ermöglichen. In diesem Kapitel werden die bereits grob definierten Anforderungen an das Datenbanksystem weiter präzisiert. Grundlage für die Präzisierung ist das mit Anwendungsfällen spezifizierte Funktionsspektrum der MeMo-Plattform. Anschließend folgt ein Überblick zu den gängigen Datenbanksystemen zur Versions- und Konfigurationskontrolle von Dateien und Dateigruppen. Eine Auswahl des Datenbanksystems für die MeMo-Plattform wird basierend auf den Anforderungen und einer kurzen Analyse getroffen. Betrachtet werden weiterhin typische Werkzeuge für die Entwicklung mechatronischer Fahrzeugsysteme (vgl. Abschnitt 4.2). Eine detaillierte Analyse dieser erfolgt jedoch nicht, da sich eine erste Implementierung nach dem am häufigsten eingesetzten Entwicklungswerkzeug richtet. Die Option, mit dem präsentierten Konzept weitere CAE-Werkzeuge anzusteuern, ist für die Implementierung zu berücksichtigen.

## 4.1 Datenbanksystem für mechatronische Fahrzeugmodelle

Das Datenbanksystem dient der strukturierten Ablage und Organisation von Modellen für mechatronische Fahrzeugsysteme. Abgelegt werden Modelle und Parametersätze jeweils als Gruppen von Dateien sowie Metadaten. Durch das Datenbanksystem soll die Möglichkeit geschaffen werden, zuvor genannte Elemente unter Versionskontrolle zu stellen. Das Konfigurationsmanagement für hierarchisch gegliederte Modelle und ein zugehöriges Variantenmanagement für die Parametersätze sind gefordert. In einem ersten Schritt werden die Anforderungen an ein Datenbanksystem präzisiert. Dieses stellt die Grundlage für die spätere Analyse (4.1.2) und die darauffolgende Entscheidung (4.1.3) dar.

### 4.1.1 Anforderungen an das Datenbanksystem

Abstrakt wurden die Anforderungen an ein Datenbanksystem zur Versions- und Konfigurationsverwaltung im Abschnitt 3.1.3 formuliert. In den folgenden Punkten werden diese separat unter Berücksichtigung der identifizierten Anwendungsfälle konkretisiert.

**Versionskontrolle für Elemente**

Ein in sich konsistentes Modell wird von einer Vielzahl von Dateien repräsentiert. Hierzu zählen beispielsweise Signalschnittstellenspezifikation, Dokumentation, Bibliotheken, Parameter, Validierungsdaten, Testdaten und ggf. auch Skripte zur automatischen Datenana-

lyse und -auswertung. Das Versionieren der einzelnen Dateien ist während der Entwicklungsphase erforderlich, um Änderungen nachzuvollziehen und im Team arbeiten zu können. Da die Dateigruppe das gesamte Modell repräsentiert, ist auch diese einer Versionskontrolle zu unterstellen. Dadurch wird sichergestellt, dass eine Gruppe von Dateien mit der exakten Version der einzelnen Dateien konsistent reproduzierbar ist. Eine Dateigruppe, die sich ausschließlich aus den darin enthaltenen Dateien zusammensetzt, stellt die kleinste versionierbare Einheit dar und trägt den Namen „Modellierungsobjekt“. In der Informatik wird die Versionierung einer Dateigruppe als Setzen einer „Baseline[40]“ bezeichnet. Auf Dateigruppenebene wird ein Berechtigungssystem benötigt, das ein Abbilden der Benutzerrollen Anwender, Entwickler, Projektleiter und Administrator (Kapitel 3.3) ermöglicht.
Um die Zusammenarbeit zwischen mehreren Entwicklern möglichst reibungsfrei zu gestalten, müssen Mechanismen vorhanden sein, die ein Arbeiten in Softwarekooperationen unterstützen. Für den Anwender muss klar erkennbar sein, ob sich ein Element derzeit in Entwicklung befindet bzw. ob es für die Verwendung durch andere Entwickler freigegeben ist. Bei Versionsmanagementsystemen werden hierzu zwei Arbeitsweisen unterschieden [Wik02]. Die Vorgehensweise nach dem Muster „Copy Modify Merge“ lässt die gleichzeitigen Änderungen von zwei Entwicklern an einer Datei zu. Anschließend werden die Dateien automatisch oder manuell zusammengeführt und in dem Datenbanksystem abgelegt. Der Vorteil an dieser Arbeitsweise liegt darin, dass Änderungen nicht im Voraus angekündigt werden müssen. Allerdings gestaltet sich ein Zusammenführen zweier unterschiedlicher Dateien (z. B. CAE-Berechnungsmodellen oder Binärdateien) als schwierig, da Änderungen durch die Modellierung in den Entwicklungswerkzeugen zustande kommen. Ein inhaltlicher Vergleich der Dateien auf ASCII[41]-Ebene lässt nur sehr schwer eine Beurteilung der Änderungen zu und verhindert so ein manuelles oder automatisches Zusammenführen.
Die zweite Vorgehensweise nach dem Muster „Lock Modify Write“ erfordert bedingt ein Sperren der Datei(en), bevor der Anwender diese verändert. Durch das Sperren hat nur ein einzelner Anwender die Möglichkeit Veränderungen vorzunehmen, die dieser anschließend in das Datenbanksystem zurückspielt und die Datei(en) wieder freigibt. Im modellbasierten Umfeld ist dieser Mechanismus von Vorteil und schafft darüber die Basis für eine Kooperation und Kommunikation im integrativen Entwicklungsprozess mechatronischer Fahrzeugsysteme. Insbesondere, wenn mehrere Entwickler an einem komplexen, hierarchisch gegliederten Modell arbeiten, das sich aus verschiedenen Teilmodellen zusammensetzt, ist eine transparente Darstellung des Entwicklungsstatus unabdingbar. Für Teilelemente und Konfigurationen ist während der Entwicklung zwischen folgenden Status zu unterschieden:

- Element wird von einem Anwender weiterentwickelt
- Private Arbeitskopie eines Entwicklers (sog. „Arbeitsversion“)
- Freigegebene Version für Anwender (sog. „Archivversion“)

[40] Eine Baseline schreibt eine Gruppe von Dateien inklusive der Metadaten auf einzelne Elemente unveränderlich und jederzeit reproduzierbar fest.

[41] ASCII : „American Standard Code for Information Interchange“

**Konfigurationsmanagement**

Innerhalb der zuvor beschriebenen versionierbaren Dateigruppen ist ebenfalls eine Abbildung hierarchischer Strukturen mechatronischer Fahrzeugsysteme erforderlich. Ein mechatronisches Fahrzeugmodell (Integrationsmodell) ist im Datenbanksystem als Konfiguration zu behandeln. Die Konfiguration hat Referenzen zu den darin verwendeten Teilelementen der unterschiedlichen Domänen. Es werden zwei Arten einer Konfiguration unterschieden, diese sind in der nachfolgenden Tabelle dargestellt.

| Bezeichnung | Beschreibung | Verwendung |
|---|---|---|
| Modul | Das Modul enthält selbst keine Dateien und stellt eine Liste von Elementen ohne eine festgehaltene Version dar. | Zusammenstellung der Elemente für die vom Entwickler benötigte Modellumgebung. |
| Modell- oder Parametersatzkonfiguration | Die Modell- oder Parametersatzkonfiguration beinhaltet selbst Dateien (Modell oder Parameter) und referenziert hierarchisch gegliederte Teilelemente mit explizit festgehaltener Version. | Reproduzierbares mechatronisches Element, das in sich konsistent ist und Teilelemente in einer spezifizierten Version enthält. Modell- oder Parametersatzkonfigurationen können selbst Bestandteil einer Konfiguration sein. |

**Tabelle 4.1- Übersicht Konfigurationen**

In Bezug auf das Modul, die Modellkonfiguration und die Parametersatzkonfiguration sind von dem Datenbanksystem folgende Anforderungen zu erfüllen:

- Die in einer Konfiguration festgehaltenen Elemente sind ohne Redundanz der Dateien festzuhalten.
- Im Falle einer Modell- oder Parametersatzkonfiguration muss eine konsistente Reproduktion des Integrationsmodells und der darin enthaltenen Teilelemente sichergestellt sein. Dies erfordert, dass die in einer Konfiguration referenzierten Elemente mit explizitem Versionsbezug der Dateigruppe festgehalten werden.
- Die in einer Modell- oder Parametersatzkonfiguration verwendeten Elemente unterstehen weiterhin einer separaten Versionskontrolle.
- Der Austausch von Teilelementen bzw. das Editieren einer Konfiguration ist zu ermöglichen.
- Um eine Reproduktion der Modell- oder Parametersatzkonfiguration zu gewährleisten ist der Status (vgl. Versionskontrolle für Elemente) zu berücksichtigen.
- Ein Transfer sowie eine Synchronisierung der Elemente zwischen dem Datenbanksystem und dem Computer des Anwenders sind unter o. g. Bedingungen zu gewährleisten.

Durch die Konfigurationen wird ein Mechanismus geschaffen, der ein simultanes Entwickeln von Teilelementen innerhalb eines Integrationsmodells in Form von Softwarekooperationen gestattet.

**Variantenmanagement**

Wie in Abschnitt 3.1.1 gefordert, sind Parametersätze als Datenvarianten zu generischen Modellen (Modellierungsobjekten) und Konfigurationen (Modellkonfigurationen) abzubilden. Zu berücksichtigen ist, dass die Parametersätze ebenfalls durch eine Dateigruppe repräsentiert werden und einer separaten Versionierung unterstellt sein müssen. Weiterhin sind Parameterreferenzen zu einem Modell versionsgenau abzubilden, da ein Parametersatz ohne ein zugehöriges Modell unbrauchbar ist. Diese Möglichkeit muss auch bestehen, wenn sich das Modell im Status „Freigegebene Version für Anwender (sog. Archivversion)“ befindet und somit festgeschrieben ist. Auf Modellierungsobjekte werden Parametersatzobjekte abgebildet. Ein Parametersatz als Datenvariante zu einer Modellkonfiguration kann als Parametersatzkonfiguration ausgeführt werden. Diese kann Referenzen auf hierarchisch gegliederte Parametersatzobjekte und Parametersatzkonfigurationen beinhalten. In diesem Fall ist zusätzlich eine Zuordnung der Teilelemente der Modellkonfiguration zu den Parametersätzen zu gewährleisten.
Abbildung 4.1 stellt die beschriebenen Anforderungen bzgl. Referenzierung von Modellen in Konfigurationen und zugehörigen Parametersätzen, bezogen auf die Granularität der Elemente im Datenbanksystem, dar.

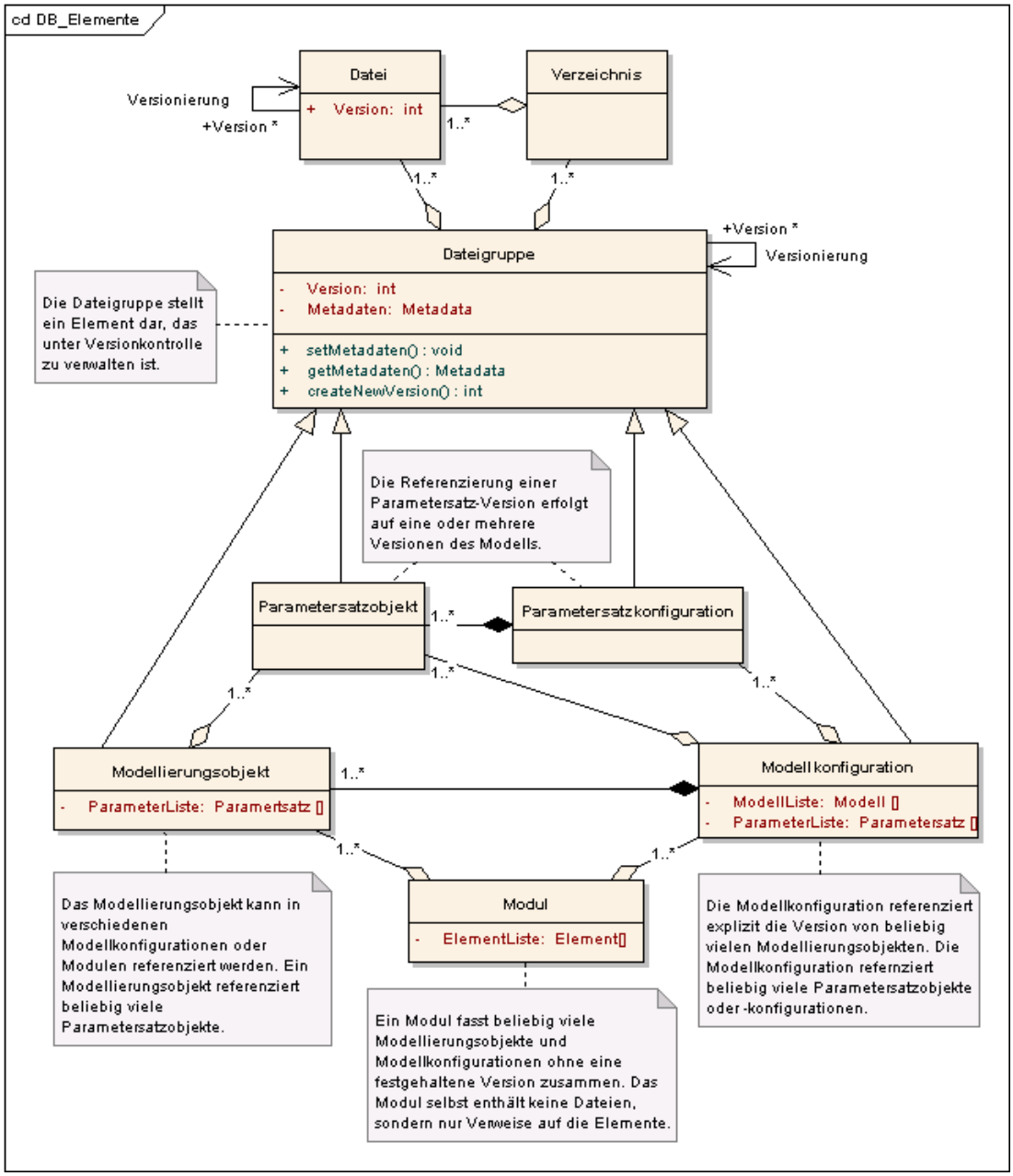


**Abbildung 4.1 - Referenzen zwischen Elementen und Versionierung von Datenbank-Elementen**

### Metadaten

Als Metadaten bezeichnet man allgemeine und eigenschaftsbeschreibende Informationen einer größeren Datensammlung. Auch im vorliegenden Fall werden Metadaten für Dateien und Dateigruppenebenen genutzt, um das Modell oder den Parametersatz kurz zu dokumentieren. Hierzu zählen unter anderem folgende Informationen:

- Kurzbeschreibung,
- Änderungsbeschreibung,
- verwendetes Entwicklungswerkzeug,

- Modellierungsgegenstand und Modellausprägung[42],
- Version der Signalschnittstellenspezifikation
- Fahrzeugprojekt
- Modellverantwortlicher etc.

Das Modell bzw. der Parametersatz wird vom Entwickler dokumentiert. Für eine zielgerichtete Anwendung helfen Metadaten zur Auswahl der Elemente im Datenbanksystem und ermöglichen ebenso eine Nachvollziehbarkeit von Änderungen an diesen. Neben den Entwicklern eines Elementes wird eine Option benötigt, mit deren Hilfe durch die Anwender der Einsatz oder Status eines Elements dokumentiert wird. Aus der Softwareentwicklung sind sog. „Tags“ bekannt, mit denen ein Element zusätzlich gekennzeichnet werden kann. Für den Einsatz von Tags im Rahmen der offenen Integrationsplattform wird hierfür folgende Struktur definiert, die auch für Elemente im Status „Freigegebene Version für Anwender (sog. Archivversion)“ vorgesehen ist:

- Tag.Name
- Tag.Autor
- Tag.Datum
- Tag.Typ
- Tag.Entwicklungsprojekt
- Tag.Kommentar

Die gesamten Metadaten sind Bestandteil eines Elements und werden auch ohne eine aktive Verbindung zum Datenbanksystem vom Entwickler bzw. Anwender zu den lokal vorhandenen Elementen benötigt und werden deshalb als lokale Kopie gehalten.

**Strukturierung**

Die Organisation von CAE-Berechnungsmodellen und zugehörigen separat versionierbaren Parametersätzen in einem zentralen Datenbanksystem erfordert zwingend eine Strukturierung des Datenbestandes. Diese kann innerhalb des Datenbanksystems mit Verzeichnissen, speziellen Elementen zur Strukturierung oder über Metadaten implementiert werden. Im vorliegen Fall ist eine Strukturierung des Datenbanksystems anhand der Komponenten mechatronischer Fahrzeugsysteme durchzuführen. Auf oberster Ebene wird zwischen Aktor, Sensor, Filter, Steuerungs- und Regelungsmodell, Streckenmodell, Fahrer & Umwelt und Gesamtfahrzeugmodell unterschieden. Ebenen zur Strukturierung des Datenbanksystems ändern sich insbesondere auf tieferer Strukturierungsebene dynamisch. Eine Änderung der Struktur und der darin abgelegten Elemente muss daher vom Datenbanksystem unterstützt werden.

[42] Modellierungsgegenstand und Modellausprägung sind Klassifizierungen aus dem hausinternen Richtlinienkatalog zur Modellierung von Matlab/Simulink-Modellen. Der „Modellierungsgegenstand“ charakterisiert ein Modell hinsichtlich seiner Zugehörigkeit zu einem Teilsystem in einem allgemeinen steuerungs- oder regelungstechnischen Gesamtsystem. Die „Modellausprägung“ charakterisiert ein Modell hinsichtlich seines Abstraktionsgrades [VWM06].

**Informationstechnische Rahmenbedingungen**

Um das Benutzer- und Rechtemanagement auf die im Datenbanksystem hinterlegten Elemente zu vereinfachen, ist eine Anbindung an die hausinterne Nutzerverwaltung wünschenswert. Eine zwingende informationstechnische Vorraussetzung ist hingegen die Möglichkeit, das Datenbanksystem über die MeMo-Plattform mit der Anwendersoftware zu verknüpfen. Die Verknüpfung muss den Transfer von Dateien sowie Metadaten unterstützen. Weiterhin sind über diese Schnittstelle eine Steuerung des Versionsmanagements und die Abbildung von Konfigurationen und Varianten erforderlich. Zukünftig ist im Sinne eines durchgängigen Prozesses eine Verknüpfung mit Anforderungsmanagementsystem sowie einem Softwarewerkzeug zur zentralen Verwaltung von Testfällen für virtuelle Funktionstests im Umfeld von MiL, SiL und HiL denkbar. Hausintern ist eine Integration des Datenbanksystems in die IT-Infrastruktur zwingend erforderlich, um eine nachhaltige Unterstützung der Anwender im Umfeld mechatronischer Fahrzeugsysteme zu gewährleisten.

### 4.1.2 Analyse gängiger Datenbanksysteme

Basierend auf den zuvor herausgearbeiteten Anforderungen erfolgt die Analyse von Datenbanksystemen und Versionskontrollsystemen. Bereits während der Analyse hat sich relativ schnell herauskristallisiert, dass relationale Datenbanksysteme[43] bzgl. der Verwaltung von Metadaten einen entscheidenden Vorteil haben, diesen jedoch beim Umgang mit Dateien und Dateicontainern mehr als einbüßen. Aus diesem Grund werden in der vorliegenden Arbeit nur die Ergebnisse für einen Teil der betrachteten Versions- und Konfigurationssysteme dargestellt. Repräsentativ wird ein System der Kategorie Versionsverwaltung, zwei Systeme der Kategorie Versions- & Konfigurationsverwaltung und ein Hinweisgewinnungssystem betrachtet. Diese sind im Einzelnen:

- CVS/Subversion (Open Source Software)[44]
- Source Integrity (MKS)
- Rational Clear Case (IBM)
- DoRIS[45] (David GmbH)

[43] Relationale Datenbanken speichern Daten in Tabellenform. Den Daten liegt ein relationales Datenmodell zugrunde. Von einem relationalen Datenmodell wird gesprochen, wenn darin Entitäten, ihre Eigenschaften und ihre Beziehungen zueinander durch das Konzept der Relation dargestellt werden. [Bal00]

[44] CVS (Concurrent Versions System) ist ein Softwaresystem zur Versionsverwaltung von Dateien. Bei fortschreitenden Versionen werden darin die Differenzen der Dateiversionen hinterlegt. Das Softwaresystem findet einen breiten Anwendungsbereich in der Quellcodeversionierung. Trotz der großen Beliebtheit hat CVS Schwächen, insbesondere gelten der Verlust der Datei-Historie bei Dateiverschiebung und die umständlichen Mechanismen zum Branching und Tagging als Nachteil gegenüber Subversion [Pre05]. Subversion ist ein frei verfügbares Open Source-Versionskontrollsystem, das Ende Februar 2004 in Version 1.0.0 verfügbar gemacht wurde [Col05]. Das Werkzeug „Subversion“ orientiert sich maßgeblich an CVS und erweitert dessen Funktionsumfang. Im Gegensatz zu CVS wurde mit Subversion versucht, einige Schwächen, insbesondere die Defizite des umständlichen Branching und Tagging, zu verbessern. Subversion ermöglicht auch ein Verschieben von Dateien, ohne dass die Historie dabei gelöscht wird. Dieser Mechanismus stellt bei Subversion die Basis für ein Branching und Tagging dar [Pre05].

[45] DoRIS: Document Information and Retrieval System

Die Ergebnisse sind in Tabelle 4.2 zusammengefasst.

| **Beurteilungskriterium** | **Datenbanksystem** | | | |
|---|---|---|---|---|
| | **CVS/Subversion (Open Source Software)** | **Source Integrity (MKS)** | **Rational Clear Case (IBM)** | **DoRIS (David)** |
| Klassifizierung des Systems | Versionsverwaltung | Versions- und Konfigurationswerkzeug | Versions- und Konfigurationswerkzeug | Hinweisgewinnungssystem |
| Literatur | [Pre05], [Col05], [Hei02], [Col04], [Sch05], [SCM06] | [VWI05], [MKS06], [Bro05] | [VWI05], [SCM06], [IBM06], [Bro05] | [Dav06] |
| Komplex: Versionskontrolle | | | | |
| Versionskontrolle für Dateien | Versionierung erfolgt nicht explizit auf die Datei, sondern auf den aktuellen Stand der Datenbanktransaktion. | Möglich | Möglich | Möglich |
| Versionskontrolle für Dateigruppen | Nur möglich durch eine Kopie der entsprechenden Dateien in ein neues Verzeichnis für jede Version. | Möglich | Über den Typ „Label“ kann auf jede Datei eine Zugehörigkeit zu einer Dateigruppe definiert werden. | Möglich |
| Branch auf eine Dateigruppe | Möglich | Möglich | Möglich | Nicht möglich. Erstellung eines neuen „Projektes“ erforderlich. Ursprung kann über ein Attribut zurückverfolgt werden. |
| Aktionen auf Dateien / Dateigruppen | Auf Verzeichnisebene möglich. | Möglich | Möglich | Möglich |
| Berechtigung auf Dateigruppen-Ebene [Ändern, Lesen, Sehen] | Nicht möglich. | Möglich [Ändern, Lesen] | Nur auf Datei-Ebene möglich [Ändern, Lesen]. | Möglich [Ändern, Lesen] |

| | | | | |
|---|---|---|---|---|
| Entwicklung im Mehrbenutzerbetrieb | CVS: Lock-Modify-Write<br>Subversion: Copy-Modify-Merge | Gegeben | Gegeben | Lock-Modify-Write |
| Lokale Statusdarstellung für die Elemente | Durch entsprechendes zusätzliches Softwaretool möglich (z. B. TortoiseCVS) | Gegeben über installierte Software auf dem Rechner. | Gegeben über installierte Software auf dem Rechner. | Nicht gegeben. |
| Komplex: Konfigurationsmanagement | | | | |
| Erstellen einer Modellkonfiguration (hierarchisches Gliedern von Teilelementen mit explizitem Versionsbezug)<br>Separate Versionskontrolle der Teilelemente ist weiterhin gegeben. | Nur möglich durch das Erstellen von Kopien des hierarchisch gegliederten Elements in einem Verzeichnis. | Möglich, indem Teilelemente in ein separates Projekt in dem Datenbanksystem referenziert werden. Dieses Projekt repräsentiert eine „Sandbox“ und stellt eine Dateigruppe für die Versionskontrolle dar. | Möglich durch sog. „Views“ mit der Nennung entsprechender „Labels“. | Möglich durch sog. „DoRIS-Links“ |
| Zusammenfassung mehrerer Dateigruppen ohne expliziten Versionsbezug (Modul) | Nicht gegeben. | Nicht gegeben. | Möglich durch sog. „Views“ mit der Nennung entsprechender „Labels“. | Nicht gegeben. |
| Redundanz der Daten | Für Konfigurationen immer redundant. | Nicht evaluiert. | Nicht gegeben. | Nicht gegeben. |
| Komplex: Variantenmanagement | | | | |
| Abbildung einer Referenz zwischen Dateigruppen | Nur möglich durch das Erzeugen eines neuen Entwicklungszweiges in den Modell und Parametersatz kopiert werden. | Nur möglich durch das Erzeugen eines neuen Entwicklungszweiges in den Modell und Parametersatz kopiert werden. | Möglich über die Definition entsprechender Attribute. | Möglich über die Definition entsprechender Attribute. |

| | | | | |
|---|---|---|---|---|
| Referenzierung auf festgeschriebene Dateigruppen | Nicht möglich. | Nicht möglich. | Nicht möglich. | Möglich über die Definition entsprechender Attribute. |
| Abbildung einer Referenz zwischen einer Modellkonfiguration und einer Parametersatzkonfiguration | Nicht möglich. | Nicht möglich. | Nicht möglich. | Möglich über die Definition entsprechender Attribute. |
| Komplex: Metadaten | | | | |
| Definition der geforderten Attribute | Nicht möglich. | Neue Attribute können erstellt werden, diese sind immer vom Datentyp „String“. | Möglich | Möglich |
| Unterliegen die Metadaten einer Versionskontrolle | Nur der Änderungskommentar | Nur die Systemattribute. Zusätzlich definierte Attribute haben keinen Versionsbezug. | Wählbar | Wählbar |
| Kennzeichnung spezieller Versionen mit Tags | Durch das Erstellen einer Kopie in eine separates Verzeichnis. Ein Tag wird beschrieben durch einen einzelnen Kommentar. | Nicht evaluiert. | Möglich | Möglich |
| Instanziierung von Attributen (z. B. Mehrfaches Anlegen von Tags auf ein festgeschriebenes Element) | Nicht möglich. | Nicht möglich. | Nicht evaluiert. | Möglich |
| Offline-Verfügbarkeit von Metadaten | Nicht gegeben. | Nicht gegeben. | Durch sog. „Snapshot-Views“ möglich. | Nicht gegeben. |

| | | | | |
|---|---|---|---|---|
| Komplex: Strukturierung | | | | |
| Strukturierung des Datenbestandes anhand mechatronischer Merkmale | Möglich | Möglich | Möglich | Möglich |
| Unterstützung einer dynamischen Struktur in dem Datenbanksystem | CVS: nein<br>Subversion: ja | Nicht evaluiert. | Nicht evaluiert. | Möglich |
| Komplex: IT-Rahmenbedingungen | | | | |
| Zugriffsmöglichkeiten durch die MeMo-Plattform | Erfordert Implementierung eines neuen lokalen Klienten. | Ein Teil des Funktionsumfangs des Systems ist über Java-EJB's[46] verfügbar. | Über Java gegeben. | Über Java-EJB's gegeben. |
| Integration in das hausinterne IT-Umfeld (z. B. Berechtigungssystem) | Nicht evaluiert. | Nicht evaluiert. | Möglich. | Aktuell gegeben. |
| Möglichkeit zur Vernetzung mit anderen hausinternen Datenbanken (z. B. Anforderungs- oder Testfallmanagement etc.) | Nicht evaluiert. | Schnittstelle zu DOORS ist vorhanden. | Schnittstelle zu DOORS ist vorhanden. | Möglich. |

**Tabelle 4.2 - Ergebnisse einer Analyse ausgewählter Datenbanksysteme**

[46] Enterprise Java Beans (EJB) sind standardisierte Komponenten innerhalb eines Java EE-Servers (Java Enterprise Edition). Sie vereinfachen die Entwicklung komplexer mehrschichtig verteilter Softwaresysteme mittels Java. Mit Enterprise Java Beans können wichtige Konzepte für Unternehmensanwendungen, z. B. Transaktions-, Namens- oder Sicherheitsdienste, umgesetzt werden, die für die Geschäftslogik einer Anwendung nötig sind. [Wik04]

### 4.1.3 Fazit: Ein Datenbanksystem für die Integrationsplattform „MeMo“

Die Analyse der unterschiedlichen Datenbanksysteme zeigt, dass keines der betrachteten alle Anforderungen erfüllt, die in Abschnitt 4.1.1 zusammengefasst sind. Vielmehr hat jedes Datenbanksystem Stärken und Schwächen. Es gilt abzuwägen, welches der Systeme den Anforderungen am nächsten kommt bzw. ob im Einzelfall die Möglichkeit besteht, durch Algorithmen in der MeMo-Plattform entsprechende Funktionalitäten zu ergänzen. Für eine Auswahl ist deshalb vorerst zu definieren, welche Kriterien von dem Datenbanksystem zwingend erfüllt werden müssen. Besonderes Augenmerk wird dabei auf die Organisation von Dateigruppen und die Abbildung der Referenzen untereinander gelegt. Unter Referenzen werden dabei die Möglichkeiten für das Konfigurations- und das Variantenmanagement mechatronischer Modelle verstanden. Die nähere Betrachtung zeigt, dass durch den flexiblen Einsatz von Attributen die Möglichkeit besteht, Anforderungen bzgl. der Abbildung von Referenzen durch Algorithmen in der MeMo-Plattform zu realisieren. Dieser Grund unterstreicht die Forderung nach einer Anbindung des Datenbanksystems an eine Middleware – die MeMo-Plattform –, wie dies im übergeordneten Konzept dargestellt ist (vgl. Abschnitt 3.2).

Die in Tabelle 4.2 dargestellten Analyseergebnisse für das Werkzeug zur Versionsverwaltung „CVS/Subversion“ zeigen deutlich, dass bereits die verfügbaren Optionen zur Versionskontrolle von Dateigruppen für die Anwendung nicht ausreichen. Das sehr häufig eingesetzte System orientiert sich maßgeblich an der Verwaltung von Quellcode, wofür eine Menge von Zusatzsoftware für unterschiedliche Programmsysteme existiert. Die unzureichenden Möglichkeiten mit Dateigruppen zu arbeiten sowie die nicht vorhandene oder praktikable Lösung für Konfigurationen oder Varianten disqualifizieren das Werkzeug für den Einsatz im erarbeiteten Konzept der Softwareumgebung. Die Gruppe der Versions- und Konfigurationswerkzeuge erweitert das Spektrum der Versionsverwaltungswerkzeuge um Optionen zur Konfigurationskontrolle. In der Regel wird dadurch das Arbeiten mit Dateigruppen wesentlich besser unterstützt. Das Konfigurationsmanagement selbst ist in den Systemen unterschiedlich realisiert. Durch zusätzlich definierte Attribute kann im Werkzeug „Rational Clear Case“ die Zuordnung von diversen Dateigruppen zu einer Konfiguration erfolgen. Der Aufbau von Modellkonfigurationen und einem Variantenmanagement für die darin enthaltenen mechatronischen Teilelemente gestaltet sich in der Praxis jedoch sehr arbeitsintensiv. Die Schwierigkeit tritt selbst bei dem Werkzeug „Source Integrity“ auf, das weiterhin nur über geringere Möglichkeiten zur Definition von Attributen verfügt. Erschwerend kommt hinzu, dass eine Kommunikation mit den Systemen eine lokale lizenzpflichtige Installation eines Client erfordert. Das System der Kategorie „Hinweisgewinnungssysteme“ wird nicht unmittelbar mit den Möglichkeiten eines Versions- und Konfigurationssystems in Verbindung gebracht. Jedoch lassen sich die Anforderungen für die Versionskontrolle von Dateien und Dateigruppen, wie gewünscht, realisieren. Das Konfigurationsmanagement bietet nur unzureichende Möglichkeiten, Dateigruppen ohne einen expliziten Versionsbezug festzuhalten. Zu Gute kommt dem System, dass Attribute frei und sehr flexibel in ihren Eigenschaften definiert werden können. Dadurch, dass die erzeugten Ele-

mente eine eindeutige Identifikationsnummer tragen, lassen sich über Metadaten durch zusätzliche Algorithmen in der MeMo-Plattform annähernd alle geforderten Anforderungen abbilden. Lediglich die Offline-Verfügbarkeit der Metadaten stellt eine größere Herausforderung dar.
Die Systeme zur Versions- und Konfigurationskontrolle spielen ihre Vorteile gegenüber den Versionsverwaltungssystemen durch einen zusätzlichen Funktionsumfang aus. Aber dennoch können sie nur schwer die wichtigsten Anforderungen erfüllen.
Das Hinweisgewinnungssystem „DoRIS“ mit der umfangreichen Möglichkeit zur Definition und Anwendung von Attributen ermöglicht im Vergleich zu den anderen Systemen noch weitere Funktionalitäten, die jedoch als separate Algorithmen in der MeMo-Plattform implementiert werden müssen. Dies gilt ebenso für Methoden zur Konfigurationskontrolle, die in Systemen wie „Rational Clear Case“ oder „Source Integrity“ bereits vorhanden sind. Aufgrund dessen, dass durch „DoRIS“ die höchste Abdeckung der Anforderungen erfolgen kann, fällt die Entscheidung für den Einsatz dieses Systems. Dies nicht zuletzt basierend auf der Tatsache, dass das System „DoRIS“ bereits in die hausinterne IT-Infrastruktur integriert ist und durch den noch andauernden Entwicklungsprozess auf zukünftige Funktionalitäten Einfluss genommen werden kann.
Aus der Kombination von dem Hinweisgewinnungssystem „DoRIS“ und den zusätzlichen Algorithmen in der MeMo-Plattform resultiert ein Funktionsumfang, mit dem die in Abschnitt 3.1 erarbeiteten Anforderungen erfüllt werden.

## 4.2 Entwicklungswerkzeug für mechatronische Fahrzeugsysteme

Während der Entwicklung mechatronischer Fahrzeugsysteme finden eine Vielzahl von Methoden und auch Werkzeuge Einsatz, die den Entwickler von der Idee bis zur Serienreife unterstützen. Der Anwender soll durch die in der vorliegenden Arbeit konzipierte offene Integrationsplattform beim Einsatz der rechnergestützten Methode und insbesondere bei der Modellbildung und -analyse unterstützt werden. Hierzu zählt ebenfalls die Funktionsabsicherung von mechatronischen Fahrzeugsystemen in einem Randbereich. Diese Entwicklungsschritte begleiten die Phasen Systementwurf, domänenspezifischer Entwurf, Systemintegration und Eigenschaftsabsicherung und greifen auf Informationen aus der Anforderungsphase zurück (vgl. Abschnitt 2.1.1). Die erzeugten Modelle und Daten sind im Sinne einer Nachvollziehbarkeit bzw. Änderungsverfolgung für das fertige Produkt abzulegen. Die Anforderungen an MeMo sind derart festgelegt, dass ein Zugriff durch Werkzeuge aus beliebigen Entwicklungsphasen möglich ist und beliebige binäre Dateien organisiert und gespeichert werden können (vgl. Abschnitte 3.1 und 4.1.1). Im Folgenden, Abschnitt 4.2.1, werden die denkbaren Werkzeuggruppen in den oben genannten Phasen betrachtet. Für die erste Implementierung von MeMo wird in Abschnitt 4.2.2 ein Werkzeug identifiziert und bzgl. der verfügbaren Schnittstellen analysiert.

### 4.2.1 Betrachtung der Entwicklungswerkzeuge im Fokus von „MeMo“

Das für die Entwicklung mechatronischer Fahrzeugsysteme herangezogene V-Modell (vgl. Abschnitt 2.1.1) wird nach der abgeschlossen Anforderungsspezifikation bis annährend zum fertigen, serienreifen Produkt durch die Modellbildung und -analyse flankiert. Beginnend mit einer ablauffähigen Spezifikation zur formalen Überprüfung auf Vollständigkeit, Widerspruchsfreiheit und der Simulation der gewünschten Eigenschaften wird eine rechnergestützte Methode eingesetzt. Nach der Erarbeitung eines domänenübergreifenden Lösungskonzeptes wird die Gesamtfunktion eines Systems in wesentliche Teilfunktionen zerlegt. Der nachfolgende domänenspezifische Entwurf setzt jeweils die darin etablierten Entwicklungswerkzeuge ein, um mit Hilfe von Berechnungen und Analysen eine detaillierte Auslegung der Teilfunktion zu erzielen. In [VDI06] werden die Entwicklungswerkzeuge der einzelnen Domänen in folgende Gruppen eingeteilt:

- CAD[47] wird zur Modellierung der Gestalt des zukünftigen Produktes verwendet. Eine Optimierung der Parameter erfolgt im Wechselspiel mit Werkzeugen der FEM[48] und der Simulation von Mehrkörpersystemen (MKS).
- Die FEM kommt zum Einsatz, um im Bereich der Strukturmechanik und -dynamik, Elektromagnetik, Fluiddynamik, Akustik oder Temperaturfelder Analysen durchzuführen.
- BEM[49] wird zur Berechnung von Anfangs- und Randwertproblemen auf den Haupteinsatzgebieten der Elektrostatik, Akustik, Hydromechanik und Thermodynamik eingesetzt.
- Simulation von Mehrkörpersystemen wird herangezogen, um das Bewegungsverhalten komplexer Systeme zu untersuchen. Diese bestehen aus einer Vielzahl gekoppelter beweglicher Teile und ermöglichen ein breites Untersuchungsspektrum, von der Bauraumberechnung mit Identifikation von Kollisionsproblemen bis hin zur Abbildung des fahrdynamischen Verhaltens von Kraftfahrzeugen.
- Fluidtechnischer Entwurf mit Hilfe von CFD[50]-Werkzeugen dient zur Analyse von thermo- und fluiddynamischen Vorgängen in abgeschlossenen Systemen wie beispielsweise der Verbrennung im Motor oder die Klimatisierung im Fahrzeug.
- Regelungstechnische Werkzeuge ermöglichen unter Einbeziehung des Streckenverhaltens eine Auslegung der Regelstrategie sowie eine Synthese und Optimierung der Parameter.
- Die geforderte Anwenderfunktion wird im Elektronikentwurf in Form von fest verdrahteter analoger oder digitaler Signalverarbeitung abgebildet und verarbeitet. Dies bedingt auch die Reaktion auf Kommando- und Sensorsignale, indem Meldungen oder Signale zur Ansteuerung von Aktoren ausgegeben werden.

[47] CAD: Computer Aided Design

[48] FEM: Finite Elemente Methode

[49] BEM: Boundary Element Method (dt. Randelementemethode)

[50] CFD: Computational fluid dynamics (dt. Strömungssimulation)

- Der Elektrikentwurf verbindet die elektrischen Komponenten eines mechatronischen Systems und stellt sicher, dass alle Komponenten mit elektrischer Energie versorgt werden.
- Der Softwareentwurf wird unterteilt in die Bereiche Betriebssystem, Laufzeitsteuerung, Hardwaretreiber und Anwendersoftware. Sichergestellt wird, dass die Anwendungssoftware auf dem Mikroprozessor in den entsprechenden Zeitintervallen arbeitet und der Zugriff auf Sensorsignale etc. gewährleistet ist.

Die aufgelisteten Werkzeuggruppen werden teilweise iterativ eingesetzt, um ein optimales mechatronisches System zu konzipieren bzw. zu entwickeln. Dies bedingt unter anderem die Systemintegration zur Untersuchung des Gesamtsystems (vgl. Abschnitt 2.1.1).
Generell besteht die Möglichkeit mit MeMo die Dateien bzw. Modelle aller o. g. Entwicklungswerkzeuge zu organisieren, da ein Ablegen in beliebigem Dateiformat möglich ist (vgl. Abschnitt 4.1.1). Hierzu zählen auch neutrale Datenformate zum Datenaustausch wie z. B. MechaStep[51] oder VHDL-AMS[52] die zwar den Austausch von Daten erleichtern, jedoch nicht von allen Werkzeugen unterstützt werden [VDI06].
Eine Festlegung eines Entwicklungswerkzeuges soll vorerst nur für eine erste Kopplung mit dem Konzept der MeMo-Plattform erfolgen. Wie in Abschnitt 3.2 erläutert ist die Architektur der MeMo-Plattform so gewählt, dass zukünftig beliebige weitere Entwicklungswerkzeuge angebunden werden können. Für eine erste Auswahl liegen folgende Kriterien zu Grunde:

- Einsatz im Bereich der Simulation mechatronischer Fahrzeugsysteme
- Aufbau von multidomänen Simulationsmodellen ggf. mit Integration von verschiedenen Entwicklungswerkzeugen
- Schnittstellen zur Kommunikation mit Werkzeugen in anderen Entwicklungsphasen wie Anforderungsmanagement und Eigenschaftsabsicherung
- Breiter hausinterner Einsatz des Entwicklungswerkzeuges

In den zuvor genannten Werkzeuggruppen für den fachspezifischen Einsatz werden diverse Werkzeuge, die teilweise in [VDI06], [Ise06], [Mey07] oder [Ast98] genannt sind eingesetzt. Beispielsweise wird ADAMS[53] für mechanische Systeme eingesetzt. DSHplus[54] kommt für hydraulische oder SPICE[55] für elektronische Systeme zum Einsatz. Modelica[56]/ Dymola[57] und Matlab/Simulink wird zur Modellierung physikalischer Systeme eingesetzt. In Matlab/Simulink existieren darüber hinaus zahlreiche Werkzeuge zur Reglerauslegung und Analyse (Simulink Control Design etc.). Aus diesen soll im nachfolgenden Abschnitt eines ausgewählt werden.

[51] MechaSTEP: Auf STEP basierendes Datenmodell zur Simulation mechatronischer Systeme.

[52] VHDL-AMS: Auf der Hardwarebeschreibungssprache VHDL basierendes standardisiertes Format zum Austausch zwischen Simulationsumgebungen unterschiedlicher Domänen.

[53] ADAMS ist ein Softwareprodukt der Firma *MSC Software Corporation*.

[54] DSHPlus ist ein Softwareprodukt der Firma *Fluidon GmbH*.

[55] SPICE wurde entwickelt von dem *Electronics Research Laboratory of the University of California, Berkeley*.

[56] Modelica ist eine Beschreibungssprache für physikalische Modelle.

[57] Dymola ist ein Softwareprodukt zur Modellierung physikalischer Systeme in der Beschreibungssprache Modelica. Dymola wird von der Firma *Dynasim* vertrieben.

### 4.2.2 Fazit: Identifikation des ersten Entwicklungswerkzeugs für „MeMo“

Ohne einen expliziten Vergleich verschiedener Entwicklungswerkzeuge durchzuführen, aber dennoch unter Berücksichtigung der in Abschnitt 4.2.1 genannten Kriterien, wird für eine erste Implementierung das Entwicklungswerkzeug Matlab/Simulink ausgewählt. Die wesentlichen Punkte für diese Entscheidung sind dabei:

- Verknüpfungsmöglichkeiten der darin aufgebauten bzw. verwendeten Modelle mit einem Anforderungskatalog in Werkzeugen, wie z. B. Doors[58].
- Matlab wird auf breiter Ebene hausintern angewendet und hat sich quasi zum Industriestandard entwickelt [Ise06].
- Methoden zur Strukturierung hierarchisch gegliederter Modelle (vgl. Abschnitt 2.1.2) werden durch verschiedene Bibliothekskonzepte unterstützt.
- Das modulare Konzept der Entwicklungsumgebung ermöglicht den Einsatz diverser Erweiterungen für die Modellierung in speziellen Anwendungsbereichen (z. B. Modellierung mechanischer Systeme mit SimMechanics[59]).
- Eine Modell- oder Toolkopplung wird durch eine Vielzahl von Werkzeugen aus unterschiedlichen Domänen angeboten. Beispielsweise durch das für Mehrkörpersimulation eingesetzte Werkzeug ADAMS [Bre04].
- Die Simulation von Multidomainsystemen, verteilt über mehrere verschiedene Computer und Programmsysteme, ist mit Hilfe zusätzlicher Software wie z. B. Exite[60] oder TISC[61] möglich.
- Für die spätere Eigenschaftsabsicherung existierten weitere zusätzliche Programmsysteme (der Firmen dSpace, Tesis, IPG etc.) um effizient mit HiL-Simulatoren zu arbeiten. Dabei ermöglicht der Real-time-Workshop[62] einen Export der Modelle aus den entsprechenden Simulatoren.

Die Kopplung des ausgewählten Entwicklungswerkzeuges mit der MeMo-Plattform erfordert gemeinsame Schnittstellen zur Datenübertragung. Im Folgenden wird darauf eingegangen, welche Schnittstellen durch das Entwicklungswerkzeug Matlab/Simulink speziell dafür bereitgestellt werden.

**Analyse der in Matlab verfügbaren Schnittstellen zur Anbindung von „MeMo“**

Für die Einbindung eines Datenbanksystems bzw. zusätzlicher Software bestehen in Matlab verschiedene Möglichkeiten. Hierzu zählen [Mat05]:

- MSCCI[63]-Schnittstelle ermöglicht eine Verbindung zu einem Versionsverwaltungssystem, in dem einzelne Dateien zur Quellcodeverwaltung abgelegt werden

[58] DOORS ist ein Softwarewerkzeug zur Anforderungsverwaltung und -spezifikation.

[59] Von der Firma *The Mathworks, Inc.* vertriebene Toolbox als Erweiterung des Funktionsumfangs in Matlab/Simulink um Modellierung mechanischer Systeme.

[60] Softwareprodukt der Firma *Extessy AG*

[61] Softwareprodukt der Firma *TLK-Thermo GmbH*

[62] Von der Firma *The Mathworks, Inc.* vertriebene Toolbox als Erweiterung des Funktionsumfangs in Matlab/Simulink um den Export der darin aufgebauten Modelle auf eine Zielplattform.

[63] MSCCI: Microsoft Source Code Control Interface

können. Eine Organisation von Modellen, wie die im Kontext mechatronischer Fahrzeugsysteme gefordert ist, ist darüber hinaus nicht möglich [Bre06].

- Eine separate „Database Toolbox“ stellt eine Datenbankschnittstelle zu relationalen Datenbanken bereit. Eine Anwendung dieser Schnittstelle ist aufgrund der benötigten Datenbankeigenschaften nicht gegeben (vgl. Abschnitt 4.1.1).
- Matlab bietet weiterhin die Möglichkeit, Fortran, C, C++, Java- und ActiveX-Elemente mit in den Funktionsumfang aufzunehmen. Ein Zugriff auf diese Elemente erfolgt mit speziellen Kommandos direkt von der Matlab-Kommandozeile.

Die von Matlab bereitgestellten Schnittstellen zu Versionskontrollsystemen über die MSCCI-Schnittstelle oder zu relationalen Datenbanken sind für die Kommunikation mit der MeMo-Plattform nicht geeignet, da ein Datentransfer sowie eine Steuerung dessen nicht wie in den Anforderungen (vgl. Abschnitte 3.1 und 4.1.1) spezifiziert möglich ist. Aufgrund dessen ist eine Schnittstelle über ein separates selbstentwickeltes Programm, das in Matlab eingebunden wird, zu realisieren. Details zur konkreten Implementierung der Kopplung zwischen der MeMo-Plattform und Matlab sind in Abschnitt 5.2 dargestellt.

# 5 Verknüpfung vorhandener Komponenten mit neuartigen Algorithmen in „MeMo“

Für das in Kapitel 3 vorgestellte Konzept einer offenen Integrationsplattform mit dem Namen MeMo sind in Kapitel 4 unterstützende Komponenten in Form eines Datenbanksystems und einem ersten CAE-Werkzeug evaluiert und ausgewählt worden. Aufgrund des im Abschnitt 3.3 beschriebenen Funktionsumfangs und dem ausgewählten Datenbanksystem ist eine Kopplung zwischen der MeMo-Plattform und dem Datenbanksystem wesentlich stärker ausgeprägt als mit dem CAE-Werkzeug. Dieses ist vielmehr als ein variabler Bestandteil des Gesamtkonzepts zu sehen. Diverse CAE-Werkzeuge, die domäneneigene Methoden für spezielle Aufgabenstellungen im Bereich der Entwicklung mechatronischer Fahrzeugsysteme mitbringen, sind mit Hilfe definierter Schnittstellen interaktiv in das Konzept (vgl. Abschnitt 3.2) einzubinden. Vor diesem Hintergrund liegt der Schwerpunkt der neuen Algorithmen, die in der MeMo-Plattform implementiert werden, in der Kooperation mit dem Datenbanksystem einerseits und einer flexiblen Schnittstelle zur interaktiven Anbindung eines beliebigen CAE-Werkzeuges andererseits. In diesem Kapitel werden diese neuen Algorithmen der MeMo-Plattform sowie eine prototypische Anbindung des Werkzeugs Matlab/Simulink beschrieben. Dieses wird stellvertretend für die im Entwicklungsprozess mechatronischer Fahrzeugsysteme zum Einsatz kommenden CAE-Werkzeuge interaktiv in das Konzept eingebunden.

## 5.1 Neue Algorithmen im Kernbaustein von „MeMo“

Das mit den Anwendungsfällen in Abschnitt 3.3 dargestellte Funktionsspektrum der MeMo-Plattform erfordert die Implementierung neuer Konzepte, um eine umfangreiche Unterstützung des Entwicklers mechatronischer Fahrzeugsysteme gewährleisten zu können. Das Makrokonzept (vgl. Abbildung 3.1) zeigt die Kopplung eines Datenbanksystems mit der MeMo-Plattform. Aufgrund der Tatsache, dass keines der betrachteten Datenbanksysteme (vgl. Abschnitt 4.1.2) die Anforderungen im Umgang mit mechatronischen Fahrzeugsystemen vollständig erfüllen kann, erfolgt die Implementierung der notwendigen Algorithmen zur Abbildung der Anforderungen in der MeMo-Plattform. Die dazu notwendigen Algorithmen werden in den nachfolgenden Abschnitten beschrieben.

### 5.1.1 Strukturierung der Daten im Datenbanksystem

Eine strukturierte Ablage der anfallenden Informationen in Form von Dateien oder Metadaten ist ein essentieller Bestandteil, um ein effektives Arbeiten und schnelles Wiederauffinden von Modellierungsprojekten gewährleisten zu können. Ausgehend von der Evaluierung verschiedener Datenbanksysteme (vgl. Abschnitt 4.1.2) ist eine Struktur für mechatronische Fahrzeugsysteme festzulegen, die eine strukturierte Ablage einzelner Modelle

und Parameter zulässt. Um die Anforderungen für das Versionsmanagement von Dateigruppen und einzelnen Dateien sowie die Dokumentation mit Hilfe von Metadaten (vgl. Abschnitt 4.1.1) zu gewährleisten, werden folgende Ebenen für die Strukturierung eingeführt:

- *Organisation der Dateien und Metadaten auf Projekt-Ebene:* Hier steht eine Strukturierung aller von Entwicklern und Zulieferern entwickelter Dateien in einer Dateigruppe im Fokus. Im Vordergrund steht für den Anwender ein schnelles Auffinden von Modellierungsprojekten aus unterschiedlichen Domänen mit Hilfe der Metadaten.
- *Organisation der Dateien innerhalb eines Projektes:* Vorschlag für eine Ablagestruktur der Dateien innerhalb einer Dateigruppe zum Speichern eines Modells oder Parametersatzes inklusive der zusätzlich benötigten Dateien und der Möglichkeit, Dateiänderungen über Metadaten zu dokumentierten.

**Organisation der Dateien und Metadaten auf Projekt-Ebene**

Für die Strukturierung auf beiden Ebenen sind die in dem Datenbanksystem verfügbaren Elemente entsprechend der nachstehenden Tabelle einzusetzen:

| Element | Verwendung |
|---|---|
| Systemknoten | Der Systemknoten wird zur Abgrenzung von anderen Applikationen, die auf das Datenbanksystem zugreifen, gewählt. Dieser stellt die höchste hierarchische Ebene dar und ermöglicht neben der Zuweisung von spezifischen Attributen das Berechtigen von Benutzergruppen im Entwicklungskontext des mechatronischen Systems (vgl. Abschnitt 5.1.3). |
| Strukturelement | Das Strukturelement wird zur Strukturierung der Daten auf Projekt-Ebene verwendet. Durch das Strukturelement werden darauf definierte Attribute hierarchisch in der darunter gegliederten Struktur, bestehend aus Strukturelementen und Projekten, vererbt. Für jedes Element können die Berechtigungen wahlweise hierarchisch vererbt oder neu definiert werden. |
| Projekt | Das Projekt repräsentiert ein Modell- oder Parametersatzelement und gruppiert die dazu gehörenden Dateien und Verzeichnisse in einem Container. Ein Projekt unterliegt inklusive der darin enthaltenen Dateien und Verzeichnisse einer separaten Versionskontrolle. Das Projekt trägt die über das Strukturelement vererbten bzw. direkt zugewiesenen Attribute. Die Berechtigungen können wahlweise hierarchisch über Strukturelemente vererbt oder über die MeMo-Plattform neu auf dem Projekt definiert werden. Der Status Archivversion bezeichnet eine festgeschriebene Version und garantiert eine Reproduzierbarkeit der darin enthaltenen Dateien, Verzeichnisse und zugewiesenen Metadaten. Als Arbeitsversion wird eine noch änderbare Version bezeichnet. |

| | |
|---|---|
| Verzeichnis | Das Verzeichnis gruppiert Dateien und weitere Verzeichnisse innerhalb eines Projekts. |
| Datei | Die Datei ist das kleinste Element im Datenbanksystem, das einer Versionskontrolle unterliegt. Im Status wird unterschieden zwischen Archivversion, Arbeitsversion und ausgecheckter Arbeitsversion. |

**Tabelle 5.1 - Elemente zur Strukturierung des Datenbestandes in DoRIS[64]**

Als Basis für den Datenbankzugriff durch die MeMo-Plattform wird ein neuer Systemknoten mit dem Namen „MeMo“ im Datenbanksystem angelegt. Für die Strukturierung der Modelle und Parameter mechatronischer Fahrzeugsysteme wird auf Strukturelemente (vgl. Tabelle 5.1) zurückgegriffen. Die zur Organisation der Elemente gewählte Struktur orientiert sich an den Komponenten mechatronischer Systeme und der zur Absicherung der Regelfunktion benötigten Streckenmodelle. Auf oberster Gliederungsebene werden die in Abbildung 5.1 gezeigten Strukturelemente angelegt.

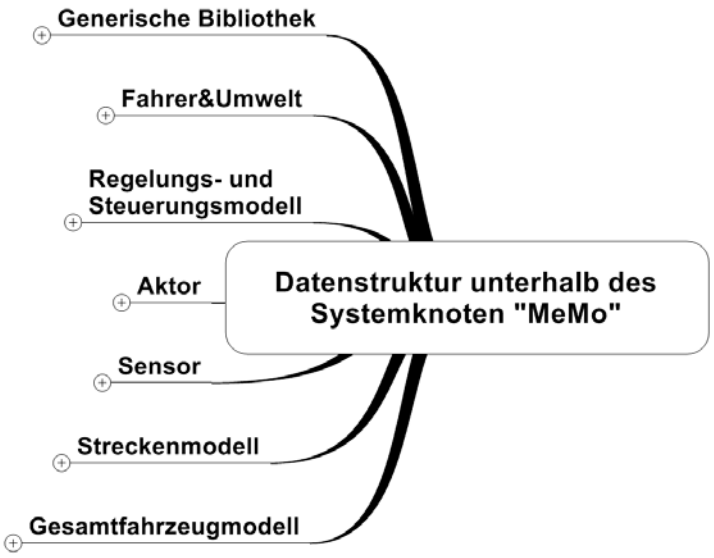


**Abbildung 5.1 - Strukturierung des Datenbestandes**

Da sich der Datenstand bzw. die Wissensbasis an verfügbaren Modellen kontinuierlich erweitert, muss die Struktur flexibel erweiterbar bzw. editierbar sein. Die MeMo-Plattform liest zur Laufzeit und nach Aufforderung durch den Nutzer, z. B. im Anwendungsfall „Element herunterladen“ (vgl. Abschnitt 3.3.1), die Datenbankstruktur aus.
Zur Ablage von neuen Dateigruppen für Modelle oder Parametersätze wird von der MeMo-Plattform auf der vom Anwender ausgewählten Ebene, in der Datenbankstruktur ein neues Projekt erzeugt, das die neuen Verzeichnisse- und Dateielemente enthält. Die Anforderungen nach separaten Versionskontrollen von Modellen, Teilmodellen und Parametersätzen (vgl. Abschnitt 4.1.1) werden von der MeMo-Plattform dadurch organisiert, dass je-

[64] Eine vollständige Übersicht aller verfügbaren Elemente und deren Eigenschaften sind in [Dav06] dokumentiert.

weils ein separates Projekt für die Organisation des Elements verwendet wird. Die Referenzen zwischen Modell und möglichen Parametersätzen werden dabei für den Anwender übersichtlich dargestellt. Aufgrund der Zeitverzögerung, die durch eine Recherche entstehen würde, um im gleichen Strukturelement zwischen Modell- und Parametersatzelement zu unterscheiden, wird ein paralleles Strukturelement für Parametersätze im Datenbanksystem angelegt. In diesem werden ausschließlich die Projekte für Parametersätze im Sinne eines Variantenmanagements gespeichert. Ein Beispiel ist in Abbildung 5.2 dargestellt. Eine Zuordnung der Abhängigkeiten erfolgt basierend auf speziell dafür definierten Attributen (vgl. Abschnitt 5.1.3). Durch die Auswertung dieser kann die MeMo-Plattform mit Hilfe der Algorithmen für das Variantenmanagement (vgl. Abschnitt 5.1.7) die Referenzen für den Nutzer auflösen und die beiden Strukturelemente zur Organisation von Modell und Parametern übereinander legen.

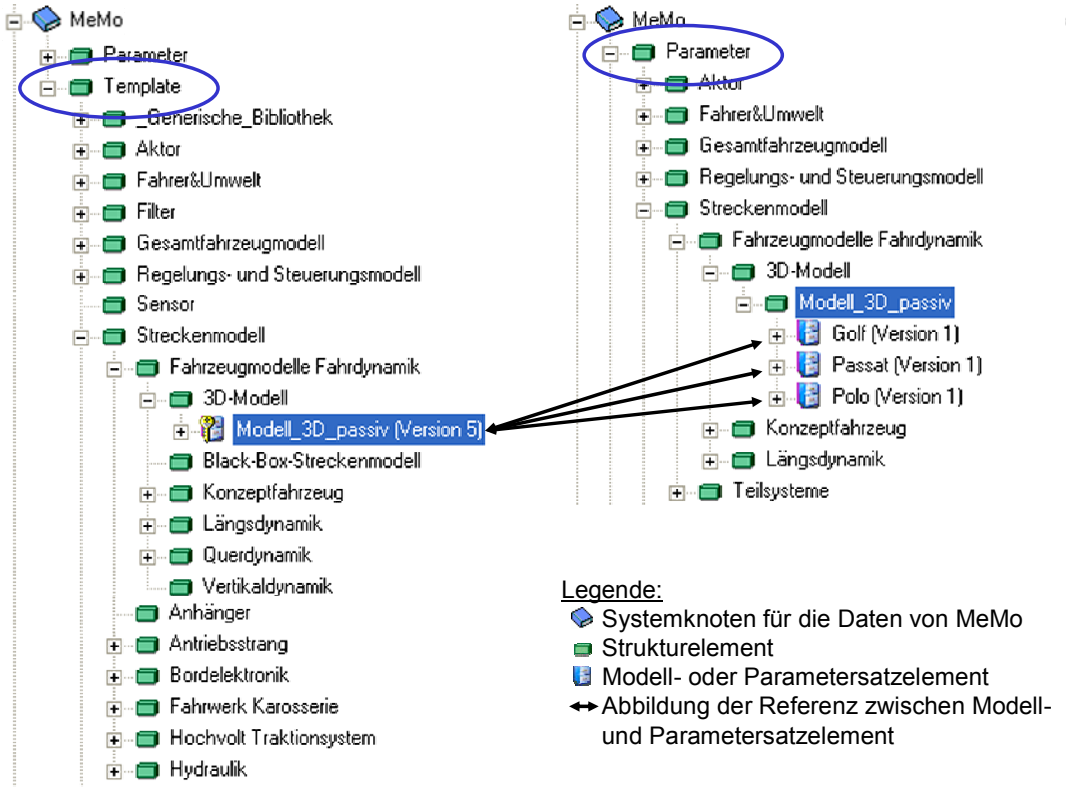


**Abbildung 5.2 - Struktur der Datenablage in MeMo**

### Organisation der Dateien innerhalb eines Projektes

Die MeMo-Plattform erzeugt für ein durch den Nutzer neu angelegtes Element in der Datenbankstruktur eine Verzeichnisstruktur als Grundlage für die Organisation der Dateien eines Modells oder Parametersatzes. In Tabelle 5.2 ist die Struktur für die beiden Elementtypen dargestellt.

| Verzeichnis | Bedeutung |
|---|---|
| **Standardstruktur für Modelle** | |
| Default-Parameter | Ablage des Parametersatzes, mit dem das Modell validiert wurde. Der Inhalt des Verzeichnisses wird durch die Interaktion mit dem CAE-Werkzeug standardmäßig mit dem Modell zur Simulation bereitgestellt. |
| Dokumentation | Dokumentation zu dem Modell |
| Konfiguration | Das Verzeichnis existiert nur bei Modellkonfigurationen (vgl. Abschnitt 5.1.4) und dient zur Ablage der Referenzen auf Teilmodelle. Durch die Algorithmen der MeMo-Plattform werden die Referenzen ohne explizite Anwenderaktion in diesem Verzeichnis abgelegt. Die im Verzeichnis enthaltenen Teilmodelle bzw. Projekte werden durch die MeMo-Plattform für die Simulation in dem CAE-Werkzeug bereitgestellt. |
| Lib | Hier werden vom Nutzer zu dem Modell benötigte Bibliotheken für die Simulation abgelegt, die keiner separaten Versionskontrolle erfordern (z. B.: Black-Box-Modelle[65]). Die darin enthaltenen Dateien werden durch die MeMo-Plattform für die Simulation in dem CAE-Werkzeug bereitgestellt. |
| Test | Unter Test erfolgt die Ablage von Stimuli- und Testdaten, die bei der Verifikation und Validierung des Modells genutzt bzw. erzeugt worden sind. |
| Zubehör | Unter Zubehör werden alle weiteren Daten abgelegt, die temporär für das Modell benötigt werden (z. B.: Skripte zur Erzeugung von Auswertungen oder Diagrammen, Auslesen von Parametern aus Messdaten etc.). |
| **Standardstruktur für Parametersätze** | |
| Dokumentation | Dokumentation zum Parametersatz. |
| Konfiguration | Das Verzeichnis existiert nur bei Parametersatzkonfigurationen (vgl. Abschnitt 5.1.7) und dient zur Ablage referenzierter Parametersätze. Die darin enthaltenen Referenzen werden für die Simulation nach Aufforderung durch den Nutzer in dem CAE-Werkzeug bereitgestellt. |
| Parameter-definition | Klassen oder Objekte zur Definition von Parametern[66] werden in diesem Verzeichnis verwaltet. |

**Tabelle 5.2 - Standardverzeichnisstruktur für Modelle und Parametersätze**

[65] Der Begriff „Black-Box-Modell“ wird für kompilierte Simulationsmodelle verwendet. Der Nutzer hat nur Zugriff auf die zuvor definierten Schnittstellen des Modells und kann in der Regel die programmierten Algorithmen sowie die hierarchische Struktur des Simulationsmodells nicht erkennen. Black-Box-Modelle werden häufig eingesetzt, um das darin Programmierte gekapselt und damit geschützt weiterzugeben, was insbesondere zwischen Automobilherstellern und Zulieferern praktiziert wird.

[66] Ein Parameterwert erfordert neben der Wertzuweisung auf eine Parametervariable einen zusätzlich spezifizierenden Text. Dieser kann beispielsweise in einer Klasse oder einem Datenobjekt festgehalten werden. Nach dem Laden der Parameter sind diese unmittelbar durch den Nutzer in der Entwicklungsumgebung verfügbar. Die Einführung von Objekten orientiert sich an dem Vorschlag der hausinternen Modellierungsrichtlinien für Matlab/Simulink/Stateflow [VWM06].

Die durch die MeMo-Plattform automatisch angelegte Standardstruktur orientiert sich an dem Vorschlag der hausinternen Richtlinie für die Modellierung von Matlab/Simulink/ Stateflow-Modellen. Diese Struktur kann vom Nutzer bis auf die Verzeichnisse „Default-Parameter“ und „Konfiguration“ beliebig modifiziert werden. Die MeMo-Plattform synchronisiert die Änderungen des Nutzers mit der im Datenbanksystem vorhandenen Struktur.

### 5.1.2 Versionskontrolle für Modelle und Parametersätze

Gefordert ist eine Versionskontrolle für einzelne Dateien und Dateigruppen (vgl. Abschnitt 4.1.1). Die Dateigruppe fasst in Form eines Elementes ein Modell oder einen Parametersatz als eine Einheit zusammen. In Abbildung 4.1 ist das Versionierungsschema für die Elemente Datei, Modellierungsobjekt und -konfiguration und Parametersatzobjekt und -konfiguration dargestellt. In MeMo sind Algorithmen zur Steuerung der Versionskontrolle auf Datei und Dateigruppenebene implementiert. Durch das Datenbanksystem wird bei Veränderung einer Datei die Versionsnummer selbstständig um einen Zähler erhöht. Die Gruppierung von Verzeichnissen und Dateien wird von der MeMo-Plattform in einem Projekt (vgl. Tabelle 5.1) gesteuert. Um ein Projekt und somit auch den Inhalt eines Projekts mit einer Versionsnummer zu versehen, wird dieses in eine sog. „Archivversion“ überführt, wie dies in Abbildung 5.3 dargestellt ist. In diesem Schritt werden die darin enthaltenen Elemente reproduzierbar festgeschrieben. Eine zukünftige Weiterentwicklung eines darin enthaltenen Elements bedingt das Erhöhen der Versionsnummer um einen Zähler und setzt das Projekt in den Status „Arbeitsversion“. Durch den Mechanismus „Lock-Modify-Write“ ist ein Bearbeiten der Datei erst dann möglich, wenn der Nutzer ein exklusives Schreibrecht auf die Dateien und somit eine „ausgecheckte[67] Arbeitsversion“ vorliegen hat. Durch die Synchronisation bzw. das Einchecken[68] eines Projektes in das Datenbanksystem werden die lokal vorgenommenen Änderungen in selbiges überführt. Mit diesem Schritt wird das exklusive Schreibrecht zurückgegeben und eröffnet anderen Entwicklern über die MeMo-Plattform die Möglichkeit, die im Projekt enthaltenen Elemente zu verändern. In diesem Status verharrt das Projekt solange, bis ein Nutzer das Projekt in den Status „Archivversion“ überführt. In jedem Status kann das Projekt für die Simulation durch berechtigte Nutzer heruntergeladen werden. Abbildung 5.3 zeigt die möglichen Optionen der Nutzer, verbunden mit dem Status eines Elements im Datenbanksystem.

[67] Durch die Aktion *auschecken* wird das exklusive Recht, Elemente zu verändern vom Datenbanksystem einem Nutzer zugeordnet.

[68] Durch die Aktion *einchecken* erfolgt eine Synchronisation veränderter Elemente mit dem Datenbanksystem. Im Anschluss wird das exklusive Recht Elemente zu verändern, zurück an das Datenbanksystem gegeben.

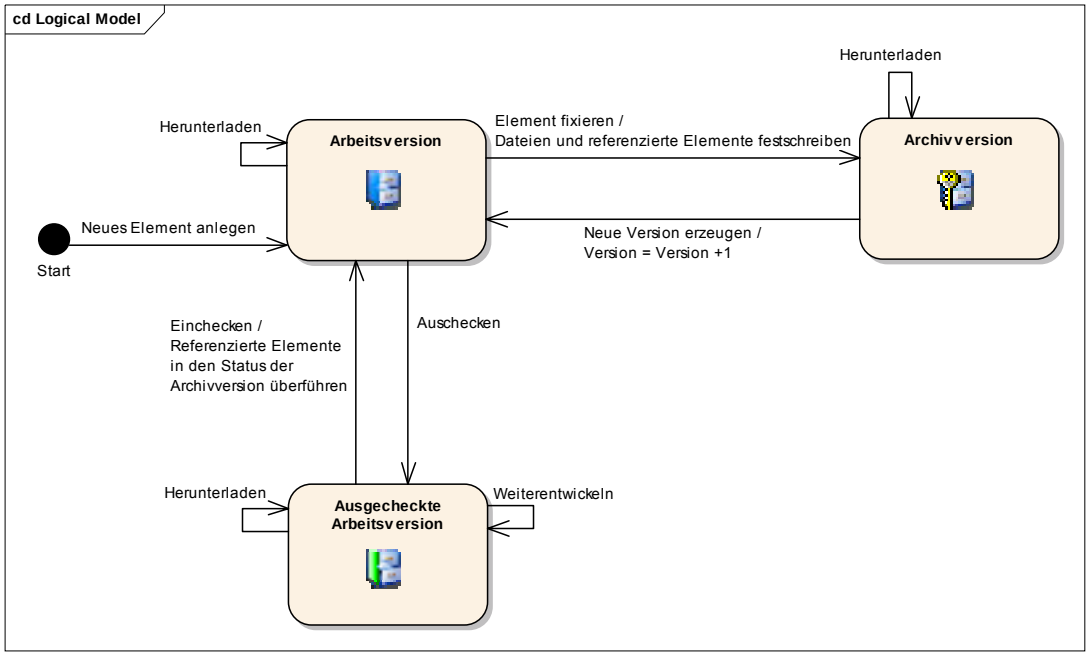


**Abbildung 5.3 - Status eines Projektes im Datenbanksystem**

Die Steuerung des Versionsmanagements wird von der MeMo-Plattform für den Nutzer abstrahiert und ermöglicht ihm somit ein Arbeiten ohne das Hintergrundwissen der Mechanismen im Datenbanksystem.

### Berechtigung auf ein Projekt

Modell- oder Parametersatz-Projekte unterliegen einem Berechtigungssystem im Datenbanksystem. Nutzer werden kontextabhängig oder nach ihrer organisatorischen Abteilungszugehörigkeit in Nutzergruppen aufgenommen. Für ein Projekt im Datenbanksystem wird für jede Nutzergruppe das Recht „Browsen“, „Lesen“ oder „Ändern“ zugeordnet. Durch die Zuordnung einer Berechtigung über eine Nutzergruppe auf ein Projekt oder ein Strukturelement wird gleichzeitig die Rolle der Nutzergruppe als Anwender oder Entwickler festgelegt (vgl. Abschnitt 3.3). Projektleiter und Administratoren werden in einer separaten Nutzergruppe zusammengefasst. Folgende Zuordnung wird angewendet:

- Nutzer mit der Rolle „Anwender“ haben mindestens ein „Lesen“-Recht auf das Projekt.
- Nutzer mit der Rolle „Entwickler“, „Projektleiter“ und „Administrator“ haben mindestens ein „Ändern“-Recht auf das Projekt.

Eine Auswertung der Rollen über die Nutzergruppen auf ein Projekt wird durch die implementierten Algorithmen der MeMo-Plattform gewährleistet, da im Datenbanksystem keine Rollenzuordnung vorgesehen ist. Die Berechtigungsstufe, die ein Lesen der beschreibenden Metadaten, aber kein Herunterladen des Projektes erlaubt, ist als neue Berechtigungs-

stufe explizit in den Algorithmen der MeMo-Plattform implementiert[69]. Zur Realisierung innerhalb der MeMo-Plattform wird eine Benutzergruppe, die alle Nutzer zusammenfasst, angelegt. Diese trägt den Namen „MeMo-Browsen“ und ist auf alle Elemente mit der Berechtigungsstufe „Ändern“ zugewiesen. Die MeMo-Plattform abstrahiert die auf einem Projekt vorhandenen Berechtigungen für den angemeldeten Nutzer und gibt diesem die für seine Berechtigungsstufe verfügbaren Aktionen frei. Für Nutzer wird nach folgenden Berechtigungsstufen unterschieden:

| Berechtigungsstufe | Beschreibung (Freigabe) |
|---|---|
| Browsen | Metadaten werden über die grafische Oberfläche der MeMo-Plattform angezeigt. Kein lesender oder schreibender Zugriff auf Dateien bzw. das Projekt. |
| Lesen | Metadaten werden dargestellt. Projekte in Archivversion können mit Hilfe der MeMo-Plattform heruntergeladen werden. Ist der angemeldete Nutzer neben der Gruppe „MeMo-Browsen“ noch in einer weiteren Nutzergruppe mit dem Erzeuger der Arbeitsversion bzw. der ausgecheckten Arbeitsversion zusammengefasst, gestatten die Algorithmen der MeMo-Plattform ebenfalls das Heruntergeladen des Projektes. |
| Ändern | Metadaten werden dargestellt. Über die MeMo-Plattform können neue Projekte im Datenbanksystem erzeugt werden. Ein Projekt in Archivversion kann mit Hilfe der MeMo-Plattform in eine Arbeitsversion (und umgekehrt) transferiert werden. Die Arbeitsversion kann ausgecheckt und weiterentwickelt bzw. verändert werden, sofern diese von einem Nutzer in der gleichen Gruppe wie der angemeldete Benutzer erstellt worden ist. Das Projekt kann in jedem Status heruntergeladen werden. |

**Tabelle 5.3 - Unterteilung der Berechtigungsstufen für Entwickler**

Wie aus Tabelle 5.3 hervorgeht, kann unter Verwendung der in der MeMo-Plattform implementierten Algorithmen durch den Projekt-Status und das Setzen von Berechtigungen eine Frei- oder Weitergabe von Elementen gesteuert werden. Ein Projekt in Archivversion repräsentiert dabei ein vollständig entwickeltes und validiertes Element. Dieses kann jederzeit exakt reproduzierbar durch die berechtigten Nutzer abgerufen werden. Eine Arbeitsversion stellt je nach Handhabung des Entwicklers einen Zwischenstand zum Test für andere Gruppenmitglieder oder eine persönliche Sicherheitskopie des Entwicklers dar. Eine Kennzeichnung bzw. die Dokumentation eines Status eines Projektes wird von der MeMo-Plattform transparent für die domänenübergreifende Zusammenarbeit angezeigt. Zusätzlich wird dies durch die im nachfolgenden Abschnitt vorgestellte Nutzung von Metadaten unterstützt.

[69] Die Anforderung, das in DoRIS bestehende Rechtesystem um ein Browsen-Recht zu erweitern, ist spezifiziert, aber aktuell noch nicht im Datenbanksystem implementiert. Der durch die MeMo-Plattform implementierte Algorithmus schafft die geforderte Funktionalität und erweitert das Rechtesystem des Datenbanksystems.

### 5.1.3 Metadaten zur Zuordnung von Merkmalen

Die Definitionen von Attributen auf Elemente im Datenbanksystem dienen der Zuordnung von Merkmalen zu einem Modell und einem Parametersatz (sog. Metadaten). Attribute werden von den Algorithmen der MeMo-Plattform eingesetzt, um die Elemente auf einer von der Datei losgetrennten Ebene zu beschreiben und Referenzen zwischen den Datenbankelementen (z. B. Modell und Parametersatz) abzubilden. Mit Attributen wie Kurzbeschreibung, Modellausprägung, Schnittstellenversion, Änderungsbeschreibung und Verantwortlicher lassen sich die vom Nutzer gesuchten Elemente zielgerichtet und effizient auffinden. Die Auswahl eines Modells kann anhand der Metadaten erfolgen und erfordert keinen Dateitransfer bzw. keine Simulation auf dem Computer des Nutzers. Das ist insbesondere dann wichtig, wenn Modelle domänenübergreifend von Nutzern eingesetzt werden sollen. Andererseits werden Attribute im Kontext der MeMo-Plattform auch zur Abbildung von Algorithmen für das Varianten- und Konfigurationsmanagement eingesetzt. Die hierfür eingesetzten Attribute mit den zugehörigen Algorithmen werden in den Abschnitten 5.1.4 und 5.1.5 im Detail beschrieben. Im Datenbanksystem DoRIS sind Attribute inklusive ihrer Eigenschaften frei definierbar. Der Datentyp kann als Wahrheitswert, Zahl, Zeichenkette und vordefinierter Listenwert vorgegeben werden. Neben dem Datentyp müssen Eigenschaften für die Attribute definiert werden. Für die Algorithmen der MeMo-Plattform wird auf folgende Eigenschaften zurückgegriffen:

- Eingabe eines Wertes vor der Überführung in den Status Archivversion (mandatory[70])
- Attributwert ist versionsabhängig
- Nutzerabhängige Eingaben
- Attributwert ist in Archivversion editierbar

Die Gruppierung von Attributen in Attributgruppen ist zulässig. Eine Attributgruppe kann in mehreren Instanzen durch die MeMo-Plattform auf einem Projekt angelegt werden, was eine mehrfache Zuordnung von Attributen mit unterschiedlichen Attributwerten ermöglicht, wie dies z. B. für die Abbildung von Tags erforderlich ist. Für die Mechanismen zur Versionskontrolle (vgl. Abschnitt 5.1.2) sind standardmäßig im Datenbanksystem folgende Attribute für jedes Element vorgegeben:

- Name des Elements
- Nutzer, der das Element zuletzt verändert hat
- Nutzer, der das Element erzeugt hat
- Änderungsdatum
- Nutzername des bearbeitenden Nutzers
- Abrufpfad auf der lokalen Festplatte des Nutzers
- Beschreibung

Die durch die MeMo-Plattform zusätzlich benötigten Attribute sind größtenteils auf Strukturelementen definiert. Die darauf definierten Attribute werden auf alle hierarchisch gegliederten Projekte, Verzeichnisse oder Dateien vererbt und können darauf jeweils mit ei-

[70] Zwingend erforderlich (engl.: mandatory) ist die im Datenbanksystem gewählte Eigenschaft für ein Attribut.

nem Wert belegt werden. Abhängig von der getragenen Information des Attributs ist weiterhin eine Zuordnung auf einzelne Elemente wie Dateien oder Verzeichnisse möglich. Die Tabelle 5.4 und die Tabelle 5.5 zeigen die definierten Attribute, die für alle durch die MeMo-Plattform organisierten Elemente eingesetzt werden. Aufgrund der unterschiedlichen Merkmale von Modellen und Parametersätzen sind weitere Attribute zur Beschreibung der Eigenschaften unerlässlich. Modellelemente werden ergänzend durch die in Tabelle 5.6 gezeigten Attribute charakterisiert. Parametersatzelemente werden durch die Attribute in Tabelle 5.7 beschrieben. Die Attributwerte werden, falls möglich, von der MeMo-Plattform ermittelt bzw. über die generierten grafischen Oberflächen von dem Nutzer abgefragt und mit den implementierten Algorithmen in die entsprechenden Datenbankfelder eingetragen.
Eine Beschreibung der Verwendung von Attributen ist in den entsprechenden Tabellen vermerkt. Außer der Attributgruppe „MeMo-Struktur“ können alle Attributgruppen in vielfacher Instanz auf die entsprechenden Elemente vergeben werden. Die Attributgruppe „Tag“ wird zur Kennzeichnung bzw. erweiterten Dokumentation von Projekten eingesetzt. Dateien und Verzeichnisse, die Bestandteil eines Tags sind, erhalten die Attributgruppe „TagPart“ mit den darin gruppierten Attributen „TagPart.Name“ und „TagPart.Tag“. Über diese Gruppe wird eine Zuordnung von ausgewählten Elementen innerhalb eines Projekts zu einem Tag auf einem Projekt realisiert. Die Attribute „Matlab-Suchpfad“ und „Matlab-Startdatei“ sind speziell für das CAE-Werkzeug Matlab/Simulink[71] implementiert und unterstützen das interaktive Arbeiten damit. Der Wahrheitswert „Matlab-Suchpfad“ steuert z. B., ob ein Verzeichnis für den Zugriff aus Matlab bereitgestellt wird. Das Attribut „Matlab-Startdatei“ spezifiziert die Initialisierungsdatei (Startdatei des Modells oder Parameters), was ein Starten der Simulation durch die Interaktion mit der MeMo-Plattform ermöglicht. Eine detaillierte Beschreibung der Attributgruppen „Parameter-Referenz“, „Modellierungsprojekt-Referenz“ und „Parameterzuordnung“ erfolgt in Verbindung mit dem Varianten- und Konfigurationsmanagement in den Abschnitten 5.1.4 und 5.1.5.

[71] Die Entwicklungsumgebung Matlab implementiert eine Variable „path“, in der alle Verzeichnispfade organisiert werden. Die in den Verzeichnispfaden verfügbaren Dateien können von Matlab gefunden und geladen werden.

| Attribut-Name | Typ | Eingabe erforderlich | Versions-abhängig | Bearbeitbar in Archivversion | Bedeutung | Anwendung auf | Anzahl mgl. Instanzen |
|---|---|---|---|---|---|---|---|
| Attribute für alle Elemente | | | | | | | |
| Änderung | mehrzeilige Zeichenkette | nein | ja | nein | Eingabe von Änderungsinformationen zur vorherigen Version | Verzeichnis, Datei | 1 |
| Anwendungshinweis | Zeichenkette | nein | ja | ja | Besonderes Kommentarfeld, das vor der Anwendung des Elements als Hinweis-Fenster für den Nutzer erscheint. Der Anwendungshinweis dient u. a. der Mitteilung von nachträglich festgestellten Fehlern in einer archivierten Projektversion. | Projekt | 1 |
| Entwicklungsplattform | Vordefiniterte Listenwerte | ja | ja | nein | Kennzeichnet die verwendete Entwicklungsplattform und Version (z.B. Matlab 7.1 oder Dymola 6.1) | Projekt | 1 |
| Kurzbeschreibung | mehrzeilige Zeichenkette | ja | ja | nein | Erläuternde Informationen zu einem Projekt. | Projekt | 1 |
| MeMo-Projektstatus | Zeichenkette | nein | ja | ja | Statusfeld eines Projekts für den Mehrbenutzerbetrieb. Festgehalten wird die Aktion und der Nutzer. | Projekt | 1 |
| Projekt-Änderung | mehrzeilige Zeichenkette | ja | ja | nein | Eingabe von Änderungsinformationen zur vorherigen Version. | Projekt | 1 |
| Projekt-Typ | Vordefinierte Listenwerte | ja | nein | nein | Definitert den Projekttyp im Kontext MeMo. Attributwerte: [Modellierungsprojekt, Parametersatz] | Projekt | 1 |
| Projekt-Konfiguration | Wahrheits-wert | ja | ja | nein | Eine Konfiguration ist ein Element, das verschiedene separat versionierbare Elemente referenziert. [true = Konfiguration; false = Objekt] | Projekt | 1 |
| Projektgegenstand | Vordefiniterte Listenwerte | ja | nein | nein | Der Modellierungsgegenstand charakterisiert ein Modell hinsichtlich seiner Zugehörigkeit zu einem Teilsystem in einem allgemeinen steuerungs- oder regelungstechnischen Gesamtsystem. Attributwerte: [Funktion, Strecke, Test, System] System kennzeichnet ein Integrationsprojekt bestehend aus verschiedenen Teilmodellen. | Projekt | 1 |
| Verantwortlich | Zeichenkette | ja | ja | nein | Verantwortlicher Entwickler des Elements. | Projekt | 1 |

**Tabelle 5.4 - Attribute zu der Beschreibung von allen Elementen (Teil 1)**

| | Attribut-Name | Typ | Eingabe erforderlich | Versions-abhängig | Bearbeitbar in Archivversion | Bedeutung | Anwendung auf | Anzahl mgl. Instanzen |
|---|---|---|---|---|---|---|---|---|
| Attribute für alle Elemente | | | | | | | | |
| Tag | Tag | Attributgruppe | | | | Kennzeichnet einen Tag auf eine Archivversion eines Projektes; aktuelle Arbeitsversion hat keine Tags. | Projekt | n |
| | Tag.Name | Zeichenkette | nein | ja | ja | Name des Tags | Projekt | 1 |
| | Tag.Autor | Zeichenkette | nein | ja | ja | Person, die den Tag angelegt hat | Projekt | 1 |
| | Tag.Datum | Datum | nein | ja | ja | Datum, wann der Tag angelegt wurde | Projekt | 1 |
| | Tag.Typ | Zeichenkette | nein | ja | ja | Typ eines Tags [Release, etc.] | Projekt | 1 |
| | Tag.Entwicklungsprojekt | Zeichenkette | nein | ja | ja | Kürzel für ein Entwicklungsprojekt | Projekt | 1 |
| | Tag.Kommentar | mehrzeilige Zeichenkette | nein | ja | ja | Mehrzeiliger Kommentar | Projekt | 1 |
| TagPart | TagPart | Attributgruppe | | | | Kennzeichnet die Zugehörigkeit eines Verzeichnis / einer Datei zu einem Tag. | Verzeichnis, Datei | n |
| | TagPart.Name | Zeichenkette | nein | ja | ja | Name eines Tags für den festgelegt wird, ob ein einzelnes Verzeichnis Tag-Bestandteil ist. Das Attribut TagPart.Name kann nur Attributwerte haben, die als "Tag.Name" definiert wurden. | Verzeichnis, Datei | 1 |
| | TagPart.Tag | Wahrheits-wert | nein | ja | ja | Beschreibt die Zugehörigkeit eines Verzeichnisses zu einem bestimmten Tag (vgl. "TagPart.Name"). | Verzeichnis, Datei | 1 |
| MeMo-Struktur | MeMo-Struktur | Attributgruppe | | | | Kennzeichnet die hierarchische Einordnung in den Strukturbaum von MeMo | Projekt | 1 |
| | MeMo-Struktur.MeMo_Ebene_1 | Vordefinierte Listenwerte mit freien Eingaben | ja | ja | ja | Erste Gliederungsebene der Datenbankstruktur. | Projekt | 1 |
| | MeMo-Struktur.MeMo_Ebene_2 | | ja | ja | ja | Zweite Gliederungsebene der Datenbankstruktur. | Projekt | 1 |
| | MeMo-Struktur.MeMo_Ebene_3 | | ja | ja | ja | Dritte Gliederungsebene der Datenbankstruktur. | Projekt | 1 |
| | MeMo-Struktur.MeMo_Ebene_4 | | ja | ja | ja | Vierte Gliederungsebene der Datenbankstruktur. | Projekt | 1 |
| | MeMo-Struktur.MeMo_Ebene_5 | | ja | ja | ja | Fünfte Gliederungsebene der Datenbankstruktur. | Projekt | 1 |
| | MeMo-Struktur.MeMo_Ebene_6 | | ja | ja | ja | Sechste Gliederungsebene der Datenbankstruktur. | Projekt | 1 |
| Spezifische Attribute für die Entwicklungsplattform (z.B. Matlab/Simulink) | | | | | | | | |
| | Matlab-Suchpfad | Boolean | nein | ja | nein | Verzeichnis soll in den Matlab-Suchpfad übernommen werden. Default = False. | Verzeichnis | 1 |
| | Matlab-Startdatei | Zeichenkette | nein | ja | nein | Enthalt den Namen der Modell-Datei, die zum Starten in der Entwicklungsumgebung benötigt wird. | Projekt | 1 |

**Tabelle 5.5 - Attribute zu der Beschreibung von allen Elementen (Teil 2)**

| | Attribut-Name | Typ | Eingabe erforderlich | Versions-abhängig | Bearbeitbar in Archivversion | Bedeutung | Anwendung auf | Anzahl mgl. Instanzen |
|---|---|---|---|---|---|---|---|---|
| Attribute ausschließlich für Modellierungsprojekte | | | | | | | | |
| | Modellierungsprojekt-ID | Zeichenkette | nein | ja | ja | Eindeutige ID (ExternalLink) des Modellierungsprojekts; Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| | Modellierungsprojekt-Name | Zeichenkette | nein | ja | ja | Name (Display-Text) des Modellierungsprojektes; Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| | MeMo-Fahrzeugprojekt | Vordefinierte Listenwerte und freie Eingaben | nein | ja | ja | Name des Fahrzeugprojekts, für das der Default-Parametersatz erstellt wurde. Das Attribut wird von der MeMo-Plattform auf das Verzeichnis "Default-Parameter" zugewiesen. | Verzeichnis | 1 |
| | Version-Schnittstelle | Vordefiniterte Listenwerte | nein | ja | nein | Version der Schnittstellenversion; Attributwerte: [unbekannt; Panama International V2; AutoSAR] | Projekt | 1 |
| | Modellausprägung | Vordefiniterte Listenwerte | ja | ja | nein | Kennzeichnet die Ausprägung eines Modells innerhalb einer mdl-Datei [physikalisch-konzeptionell, verhaltensorientiert, implementierungsorientiert]. | Datei | 1 |
| Parameter-Referenz | Parameter-Referenz | Attributgruppe | | | | Dient zur Referenzierung von Parametersätzen (spez. DoRIS-Projekt), die mit diesem Modell verwendet werden können. | Projekt | n |
| | Parametersatz-ID | Zeichenkette | nein | ja | ja | Eindeutige ID (ExternalLink) des Parametersatzes (siehe Attribut Parametersatz-ID für Parametersätze); Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| | Parametersatz-Name | Zeichenkette | nein | ja | ja | Name (Display-Text) des Parametersatzes (siehe Attribut Parametersatz-Name für Parametersätze); Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| | MeMo-Fahrzeugprojekt | Zeichenkette | nein | ja | ja | Name des Fahrzeugprojekts für das der Parametersatz erstellt wurde (siehe Attribut MeMo-Fahrzeugprojekt für Parametersätze - Definition auf allg. Ebene); Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |

**Tabelle 5.6 - Attribute zu der Beschreibung von Modellelementen**

| | Attribut-Name | Typ | Eingabe erforderlich | Versions-abhängig | Bearbeitbar in Archivversion | Bedeutung | Anwendung auf | Anzahl mgl. Instanzen |
|---|---|---|---|---|---|---|---|---|
| Attribute ausschließlich für Projekte mit Parametersätzen | | | | | | | | |
| | Parametersatz-ID | Zeichenkette | nein | ja | ja | Eindeutige ID (ExternalLink) des Parametersatz-Projektes; Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| | Parametersatz-Name | Zeichenkette | nein | ja | ja | Display-Text eines Projektes; Format: <<Projektname (Version X)>>; Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| Modellierungsprojekt-Referenz | Modellierungsprojekt-Referenz | Attributgruppe | | | | Dient der Referenzierung von Modellierungsprojekten (spez. DoRIS-Projekt), die diesen Parametersatz verwenden. | Projekt | n |
| | Modellierungsprojekt-ID | Zeichenkette | nein | ja | ja | Eindeutige ID (ExternalLink) des Modellierungsprojektes (siehe Attribut Modellierungsprojekt-ID für Modellierungsprojekte); Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| | Modellierungsprojekt-Name | Zeichenkette | nein | ja | ja | Name (Display-Text) des Modellierungsprojekts (siehe Attribut Modellierungsprojekt-Name für Modellierungsprojekte); Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| Parameterzuordnung | Parameterzuordnung | Attributgruppe | | | | Dient zur Zuordnung von hierarchisch gegliederten Parameterprojekten zu den hierarchisch gegliederten Modellierungsprojekten einer Modellkonfiguration; Attributgruppe wird von der JAVA-API auf Parameterkonfigurationen erzeugt. | Projekt | n |
| | Parameterzuordnung. Modellierungsprojekt-ID | Zeichenkette | nein | ja | ja | Eindeutige ID (ExternalLink) des Modellierungsprojekts dient zur Zuordnung der Attributgruppe zu einer Modellierungsprojekt-Referenz; Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| | Parameterzuordnung. Parameter-ID | Zeichenkette | nein | ja | ja | Eindeutige ID (ExternalLink) eines hierarchisch geschachelten Parameterprojekts dient zur Zuordnung des Parameterprojekts in dieser Instanz; Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |
| | Parameterzuordnung. Modell-ID | Zeichenkette | nein | ja | ja | Eindeutige ID (ExternalLink) eines hierarchisch geschachelten Modellierungsprojekts dient zur Zuordnung des Modellierungsprojekts in dieser Instanz; Attributwert wird über die JAVA-API gesetzt. | Projekt | 1 |

**Tabelle 5.7 - Attribute zu der Beschreibung von Parameterelementen**

**Vererbungshierarchie von Attributen**

Neben den allgemein beschreibenden Attributen für die Elemente werden zusätzlich für die Modelle in den unterschiedlichen Domänen physikalisch spezifische Attribute zur Beschreibung angeboten. Mit steigender Hierarchietiefe addieren sich die auf den Strukturelementen definierten und hierarchisch vererbten Attribute und beschreiben die Teilkomponenten mechatronischer Fahrzeugsysteme in ihren physikalischen Eigenschaften detailliert, sofern durch den Nutzer über die MeMo-Plattform ein Attributwert vergeben wurde.

### 5.1.4 Abbildung des Konfigurationsmanagements

Ein Konfigurationsmanagement wird für mechatronische Fahrzeugmodelle genau dann benötigt, wenn sich ein Komplettsystem aus mehreren unterschiedlichen und ggf. domänenspezifischen Teilelementen zusammensetzt. Arbeiten darüber hinaus mehrere Entwickler an verschiedenen Teilelementen, ist sowohl die Versionskontrolle dieser als auch die Konfiguration der Teilelemente zu einem Ganzen erforderlich. Aufgebaute und validierte Konfigurationen als mechatronische Systeme können in übergeordneten Regelsystemen des Fahrzeugs eingesetzt werden und müssen aus diesem Grund stets so reproduziert werden können, wie sie nach erfolgreichem Testen abgelegt worden sind (vgl. Abschnitt 3.1.2).
Abhängig vom Entwicklungsprojekt und dessen Stadium ist zu unterscheiden, ob eine Gruppierung der Elemente eines mechatronischen Systems mit oder ohne Versionsbezug erfolgt. Eine Gruppierung ohne Versionsbezug in einem Modul eignet sich, wenn mehrere Entwickler an einer größeren Anzahl von Teilelementen arbeiten, die sich dynamisch verändern und über zuvor im Projekt fest abgestimmte Schnittstellen Werte ausgetauscht werden. Da eine Reproduktion des Moduls im Falle einer Aktualisierung der enthaltenen Elemente nicht gewährleistet wird, liegt der Fokus auf Modellen, die in keiner mathematischen Kopplung miteinander verbunden sind. Im Gegensatz dazu werden Teilelemente inklusive ihrer Version in einer Modellkonfiguration festgehalten. Dies garantiert eine Reproduktion der Konfiguration zu jedem Zeitpunkt und ermöglicht somit den Einsatz in hierarchisch übergeordneten Systemen. Die Weiterentwicklung darin enthaltener Teilelemente kann schrittweise durchgeführt werden. Dadurch erfolgt dann auch die stufenweise Verifizierung der neuen Version von Teilmodellen, bevor eine neue Version der Modellkonfiguration abgelegt wird.
Eine tabellarische Auflistung von Unterscheidungsmerkmalen zwischen Modul und Konfiguration ist in der nachstehenden Tabelle 5.8 zusammengefasst.

| Merkmal | Modul | Konfiguration |
|---|---|---|
| Fokus/Anwendung | Organisation von Teilelementen für die Zusammenarbeit mehrerer Entwickler. Mengenoperationen für die enthaltenen Projekte (Bsp.: Tag anlegen) sind möglich. Das Modul findet Anwendung, | Versionsgenaue Organisation von Teilelementen eines Gesamtsystems, dessen Reproduktion sichergestellt ist, durch den Systemverantwortlichen. Die Konfiguration |

| | | |
|---|---|---|
| | wenn die Teilelemente einer parallelen und dynamischen Änderung durch mehrere Entwickler unterliegen. | wird eingesetzt, um das Gesamtsystem stets reproduzieren zu können und eine sporadische parallele Entwicklung durch mehrere Entwickler zu ermöglichen. |
| Gruppierung der Elemente | Modell<br>Parametersatz | Modellierungsobjekt und Modellkonfiguration oder Parametersatzobjekt und Parametersatzkonfiguration |
| Identifikation der festgehaltenen Elemente | Elemente werden ohne Versionsbezug festgehalten. | Explizites Festhalten der Teilelemente inkl. der verwendeten Version. |
| Inhalt des Gruppierungs-Elements | Ein Modul ermöglicht nur eine Gruppierung verschiedener Projekte. Ein Ergänzen von Dateien ist nicht möglich. | Modellkonfigurationen können selbst Dateien, Verzeichnisse oder Modelle enthalten. |
| Metadaten | Das Modul trägt keine Metadaten. | Metadaten wie in Abschnitt 5.1.3 vorhanden. |
| Versionierung | Die in einem Modul enthaltenen Verweise auf Modelle, Parameter oder Modellkonfigurationen werden versionslos in einer XML-Datei hinterlegt. Die XML-Datei steht unter Versionskontrolle. Ein Speichern zusätzlicher Dateien auf Modulebene ist nicht möglich. | Die Modellkonfiguration wird zusammen mit den darin enthaltenen Dateien und separat versionierbaren Teilmodellen festgeschrieben. |
| Reproduktion der Elementgruppe | Reproduktion basiert auf Tag-Informationen oder zur Laufzeit selektierten Projekt-Versionen. Eine exakte Reproduktion kann nicht gewährleistet werden. | Eine exakte Reproduktion der Modellkonfiguration ist möglich. |
| Datenvarianten | Kein Variantenmanagement | Datenvarianten 1:n |
| Nutzung | Softwarekooperationen | Aufbau hierarchisch geschachtelter Modelle und Softwarekooperationen |

**Tabelle 5.8 - Differenzen zwischen Modul und Modellkonfiguration**

Sowohl das Modul als auch die Modellkonfiguration wird dem Nutzer durch die Algorithmen der MeMo-Plattform bereitgestellt. Im Folgenden werden diese getrennt für das Modul und die Modellkonfiguration beschrieben.

**Modul**

Das Modul stellt eine flache Liste von Modellierungsobjekten oder -konfigurationen dar. Eine Erstellung oder ein Bearbeiten eines Modul erfolgt über die grafischen Oberflächen der MeMo-Plattform. Im Hintergrund wird durch die Algorithmen eine XML-Datei in dem Datenbanksystem erzeugt. Der Einsatz eines Moduls erfordert vor der Anwendung bzw. dem Herunterladen eine Spezifikation der darin enthaltenen Elemente in ihrer Version, wie dies z. B. für ein mechatronisches Lenkungsmodell in Abbildung 5.4 zu sehen ist.

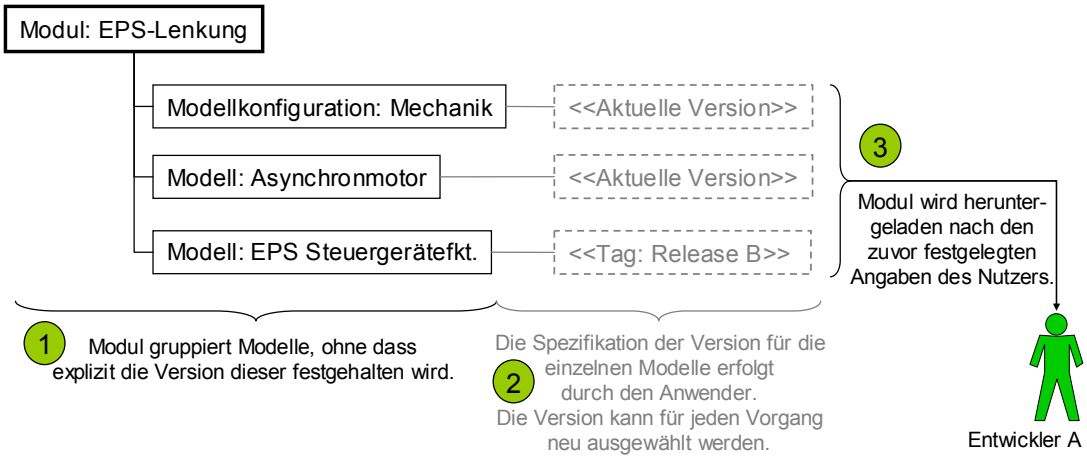


**Abbildung 5.4 - Anwendung eines Moduls am Beispiel der EPS-Lenkung**

Basis für die Auswahl ist entweder die Version oder ein Tag, der ggf. auf einzelne oder alle Elemente des Moduls gesetzt ist. Die Zusammenstellung der Versionen von Elementen in einem Modul wird auf die Festplatte des Anwenders geladen und unterliegt nicht der Versionskontrolle. Das Modul selbst mit den darin gruppierten Elementen wird in der Datenbank gespeichert. Die dazu angelegte XML-Datei und somit die Elementliste (ohne festgehaltene Version) unterliegt der Versionskontrolle. In der XML-Datei werden durch die MeMo-Plattform die Elemente mit einer eindeutigen Identifikationsnummer festgehalten, wie dies in Abbildung 5.5 z. B. für ein Fahrzeugdynamikmodell mit EPS dargestellt ist.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <ns1:DE.VWAG.MODELLBIBLIOTHEK.WEBSERVICES
    xmlns:ns1="http://www.tempuri.org/WebServices.xsd">
    <ns1:VERSION>1</ns1:VERSION>
  - <ns1:MODULEPROJECTCOLLECTION>
    - <ns1:MODULEPROJECT ns1:name="ESM_Fahrdynamikmodell_EPS">
        <ns1:EXTERNALLINK>doris://1067543</ns1:EXTERNALLINK>
      </ns1:MODULEPROJECT>
    - <ns1:MODULEPROJECT ns1:name="EPS_Lenkung">
        <ns1:EXTERNALLINK>doris://1046421</ns1:EXTERNALLINK>
      </ns1:MODULEPROJECT>
    - <ns1:MODULEPROJECT ns1:name="Asynchronmotor_EPS">
        <ns1:EXTERNALLINK>doris://1045411</ns1:EXTERNALLINK>
      </ns1:MODULEPROJECT>
    - <ns1:MODULEPROJECT ns1:name="EPS_ECU">
        <ns1:EXTERNALLINK>doris://576836</ns1:EXTERNALLINK>
      </ns1:MODULEPROJECT>
    - <ns1:MODULEPROJECT ns1:name="Lenkmechanik">
        <ns1:EXTERNALLINK>doris://894556</ns1:EXTERNALLINK>
      </ns1:MODULEPROJECT>
    </ns1:MODULEPROJECTCOLLECTION>
</ns1:DE.VWAG.MODELLBIBLIOTHEK.WEBSERVICES>
```

**Abbildung 5.5 - Beispielhafter Inhalt der XML-Datei eines Moduls**

Die Identifikationsnummer („ExternalLink“) eines Projekts im Datenbanksystem DoRIS verweist immer auf ein Projekt in einer speziellen Version[72]. Um dennoch ein Modul abzubilden, wird für alle enthaltenen Elemente die Identifikationsnummer des Projekts in der ersten Version eingetragen. Die MeMo-Plattform recherchiert nach den vorhandenen Versionen und Tags eines Elements, bevor diese aufbereitet für den Nutzer zur Spezifikation dargestellt werden. Nach der Spezifikation der Teilelemente in Version/Tag werden diese auf die lokale Festplatte transferiert. Das Modul bleibt weiterhin als Gruppierungselement vorhanden und ermöglicht Mengenoperationen wie, z. B. einchecken, auschecken, Tag anlegen oder Interaktionen mit dem CAE-Werkzeug.

### Modellkonfiguration

Die Modellkonfiguration wird zum Aufbau komplexer hierarchisch gegliederter Simulationsmodelle genutzt und lässt, anders als das Modul, eine hierarchische Strukturierung der Elemente zu. Die Anwendung von Modellkonfigurationen empfiehlt sich immer dann, wenn aufgebaute Konfigurationen auf übergeordneter Ebene wiederverwendet werden und nur sporadisch eine Weiterentwicklung der Teilelemente vorgenommen wird. Da die Modellkonfiguration selbst Dateien und Verzeichnisse beinhalten kann, wird zumeist ein Integrationsprojekt abgebildet, das andere Modellierungsobjekte oder -konfigurationen referenziert. Die separate Versionierung der referenzierten Elemente bleibt dabei unangetastet. Vielmehr werden für den Nutzer Elemente in einer neuen Version farblich gekennzeichnet und können schrittweise nach der Durchführung eines Integrationstests aktuali-

[72] Das Datenbanksystem DoRIS kennt keine eindeutige Identifikationsnummer auf eine Versionslinie eines Projektes.

siert werden. Im Gegensatz zum Modul werden die verwendeten Referenzen mit der Versionsnummer (der Archivversion) festgehalten und garantieren so eine Reproduktion eines einmal abgelegten Entwicklungsstandes. Für die Verwendung in hierarchisch übergeordneten Konfigurationen stellt die Reproduktion ein wichtiges Kriterium dar, ohne das die Konsistenz der Modelle nicht gewährleistet werden kann. Die MeMo-Plattform unterstützt den Nutzer während der Arbeit mit Modellkonfigurationen durch grafische Oberflächen und steuert im Hintergrund die entsprechenden Methoden im Datenbanksystem zur Abbildung der hierarchischen Konfigurationen. Für die Implementierung wird auf bidirektionale Verweise zurückgegriffen. Es können somit alle Daten der verwendeten Teilelemente innerhalb der Konfiguration vollständig abgebildet werden, ohne dass datenbankintern Kopien erzeugt werden müssen. Die Verweise werden von der MeMo-Plattform in der Modell- oder Parametersatzkonfiguration in dem Verzeichnis „Konfiguration“[73] abgelegt. Beispielhaft für die EPS-Lenkung ist in Abbildung 5.6 gezeigt, wie diese aus den Modellierungsobjekten Asynchronmotor_EPS, EPS_ECU und der Lenkmechanik aufgebaut ist.

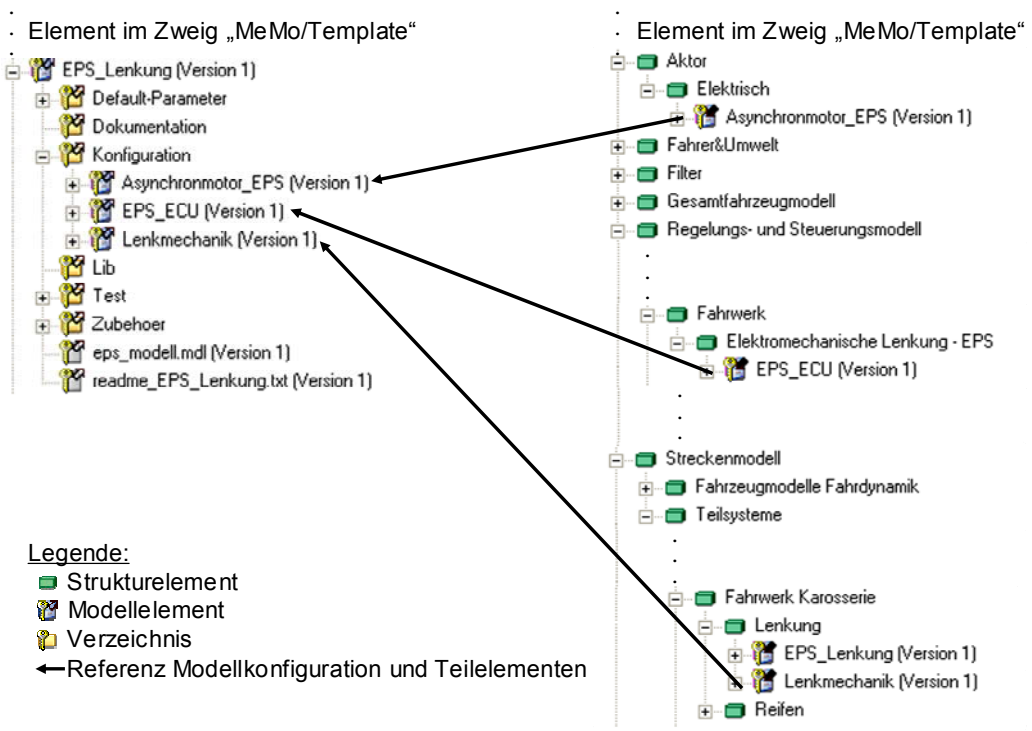


**Abbildung 5.6 - Aufbau von Modellkonfigurationen mit bidirektionalen Verweisen**

Liegt die Konfiguration als Arbeitsversion vor, können darin durch die MeMo-Plattform Elemente in Arbeitsversion referenziert werden. Sobald die Konfiguration in den Status Archivversion überführt wird, müssen die enthaltenen Elemente ebenfalls auf eine Archivversion verweisen, um letztendlich die Konfiguration mit der Zusammenstellung an Dateien,

[73] Das Verzeichnis „Konfiguration“ ist als Verzeichnis für Modell- oder Parametersatzkonfiguration vorgesehen, um Referenzen zu Teilelementen abzulegen (vgl. Abschnitt 5.1.1).

Verzeichnissen und zusätzlich referenzierten Projekten festzuschreiben. Die Tatsache, dass für die Abbildung der Konfiguration bidirektionale Verweise genutzt werden, ermöglicht darüber hinaus eine Recherche zu der Verwendung eines einzelnen Modellierungsobjekts oder einer Modellkonfiguration. Dadurch wird die Möglichkeit geschaffen, den Nutzer beim Aufbau einer neuen Konfiguration mit dem a-priori-Wissen im Datenbanksystem (der Wissensbasis) zu unterstützen. Sobald in einer Modellkonfiguration ein Teilelement vorhanden ist, wird durch die MeMo-Plattform ausgewertet und angezeigt, welche Elemente in der Vergangenheit mit den bereits vorhandenen verbaut waren. Dies ermöglicht den effektiven Aufbau neuer oder ähnlicher Strukturen mit heterogenen Teilelementen, die sehr wahrscheinlich[74] bzgl. ihrer Schnittstellen und Inhalte zusammenpassen.

### 5.1.5 Abbildung integrativer Entwicklungsprozesse

Die Entwicklung mechatronischer Fahrzeugsysteme erfordert aufgrund ihres interdisziplinären Charakters integrative Entwicklungsprozesse, durch die Algorithmen der MeMo-Plattform abgebildet werden können. Basis hierfür stellen die zuvor beschriebenen Konfigurationselemente (Modul und Modellkonfiguration) dar. Die in einem Modul oder einer Modellkonfiguration zusammengefassten Elemente stehen weiterhin unter separater Versionskontrolle und können durch die berechtigten Nutzer weiterentwickelt werden. Ein mechatronisches Fahrzeugsystem, das durch eine Modellkonfiguration abgebildet ist, ermöglicht das parallele Arbeiten mehrerer Entwickler in einer heterogenen Modelllandschaft mit unterschiedlichen Verantwortungsbereichen. Abbildung 5.7 zeigt beispielhaft einen Prozess mit drei beteiligen Entwicklern.
Die Entwickler B.1 und B.2 arbeiten parallel an separat versionierbaren Elementen, die übergeordnet in einem Integrationsprojekt durch den Entwickler A zusammengeführt werden. Der Entwickler B.2 arbeitet dabei selbst an einer Konfiguration, die aus verschiedenen Teilelementen aufgebaut ist. Aufgrund dessen, dass die Modellkonfiguration inklusive der darin enthaltenen Teilelemente bei den Entwicklern vorliegt und im Falle der Weiterentwicklung mit der zentralen Datenbank synchronisiert wird, wird durch die MeMo-Plattform automatisch signalisiert, ob eine neue Arbeits- oder Archivversion vorliegt. Dies garantiert, dass der aktuelle Entwicklungsstand durch alle im Projekt beteiligten Entwickler eingesehen werden kann. Aktualisierte Elemente lassen sich durch die in der MeMo-Plattform implementierte Funktionalität schrittweise aktualisieren und testen.

[74] Eine Unsicherheit bleibt auf Grund dessen bestehen, dass nicht garantiert werden kann, dass bereit in der Vergangenheit aufgebaute Konfigurationen bzgl. ihrer Schnittstellen harmonieren.

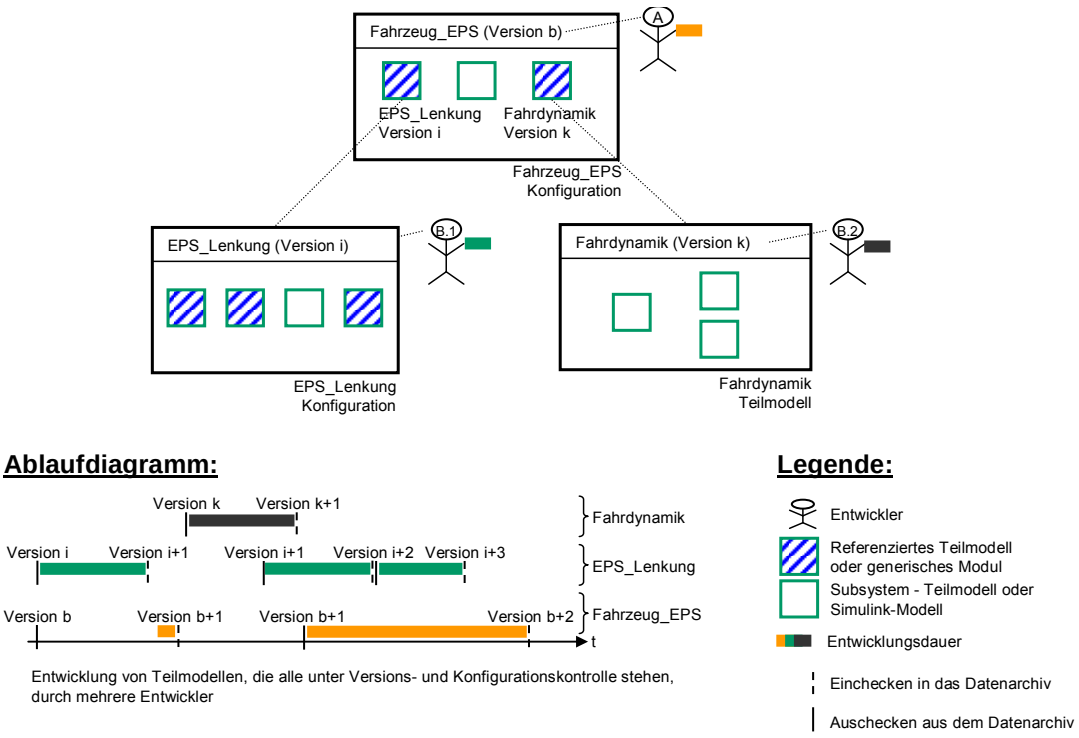


**Abbildung 5.7 - Paralleles Arbeiten an einer Modellkonfiguration**

### Webportal zur Integration von Zulieferern

Neben den internen Entwicklern unterstützen oftmals auch externe Lieferanten die Entwicklung mechatronischer Fahrzeugsysteme. Im Verantwortungsbereich der Lieferanten steht häufig die Implementierung einer Steuergerätefunktion, die vom Funktionsverantwortlichen spezifiziert und beauftragt wird. Die Lieferanten benötigen dazu vielfach eine Abbildung des Streckenverhaltens, das vom Funktionsverantwortlichen neben der Spezifikation der Steuergerätefunktion bereitgestellt wird. Die MeMo-Plattform ermöglicht durch spezielle Algorithmen den Export von Elementen aus der Datenbank. Die exportierten Elemente werden zusammen mit einer XML-Datei an den Lieferanten weitergegeben. Die durch den Lieferanten erzeugte Version wird basierend auf den Informationen in der zuvor exportierten XML-Datei in das Datenbanksystem importiert und steht nach der Freigabe des Funktionsverantwortlichen im Datenbanksystem zur Anwendung bereit. Abbildung 5.8 zeigt das Vorgehen anhand eines Ablaufcharts.

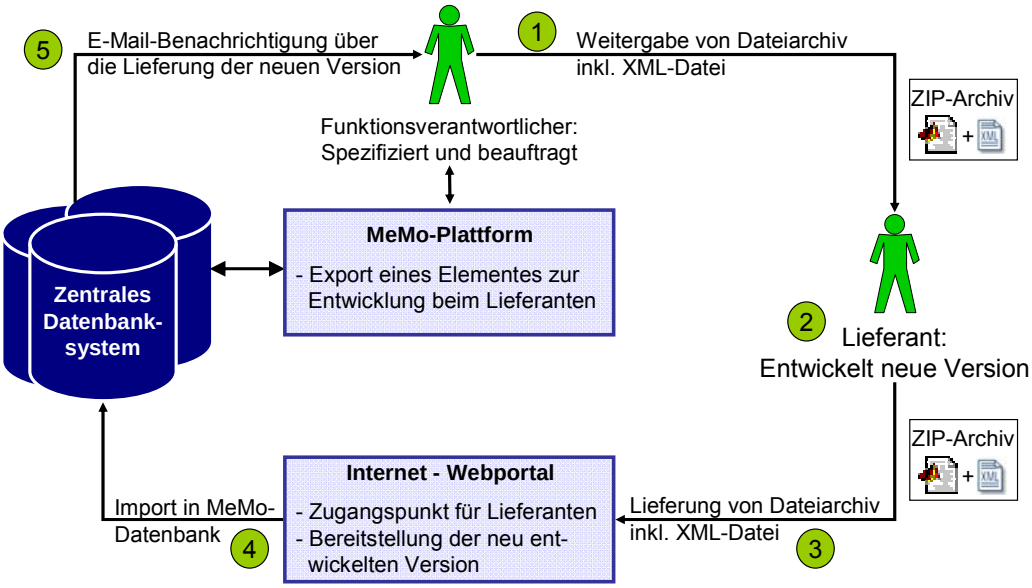


**Abbildung 5.8 - Integration von Lieferanten (Ablaufdiagramm)**

Die Steuerung des Imports sowie die damit ausgelösten Aktionen (z. B. der Versand einer E-Mail) erfolgt aufgrund der bereitgestellten XML-Datei. Diese enthält die zukünftigen Metadaten, spezifiziert das Projekt, auf das importiert wird und legt fest, welche Anwender nach dem Import zusätzlich mit einer E-Mail benachrichtigt werden. Das Webportal zur Integration von Lieferanten sowie das Modul und die Modellkonfiguration ermöglichen über die MeMo-Plattform ein transparentes Arbeiten an einem mechatronischen Fahrzeugsystem. Durch die Statusanzeige der einzelnen Elemente wird sofort nach Änderung eines Elementes der aktuelle Status für alle Anwender ersichtlich.

### 5.1.6 Spezifikation der Modellschnittstellen

Beim Aufbau neuer Gesamtfahrzeugmodelle wird oftmals auf bereits bestehende Teilmodelle zurückgegriffen, die in dem Datenbanksystem mit der MeMo-Plattform verwaltet werden. Der Aufbau neuer Modelle kann in Simulink erfolgen, indem Modellschnittstellen[75] unterschiedlicher Teilmodelle miteinander verknüpft werden. Insbesondere, wenn der Aufbau aus Teilmodellen erfolgt, die von unterschiedlichen Entwicklern bereitgestellt worden sind, erfordert dies eine präzise Kenntnis der über die Modellschnittstellen übertragenen Daten, um letztendlich die richtigen Ein- und Ausgänge miteinander zu verbinden. Ein in Simulink aufgebautes Modell ermöglicht die Implementierung von Modellschnittstellen,

[75] Als Modellschnittstellen werden die für den Datenfluss vorgesehenen Ein- und Ausgänge eines Modells bezeichnet, die dazu dienen, ein Signal an ein anderes Modell weiterzugeben. Typischerweise werden in Simulink die auf oberster Hierarchieebene vorhandenen Ein- und Ausgänge mit anderen Modellschnittstellen zur Datenweitergabe verknüpft.

die frei vom Entwickler definierte Signale übertragen können[76]. Simulink stellt eine Möglichkeit zur Dokumentation der Schnittstellen über die Eigenschaften Datentyp, Abtastrate und Signaltyp [real, komplex] zur Verfügung [Mat05]. Für eine ganzheitliche Dokumentation sind diese Eigenschaften unzureichend und müssen deshalb oftmals in zusätzlichen Textdokumenten ergänzt werden. Um dem Entwickler die benötigten Informationen direkt im Modell anzubieten und diese darüber hinaus auch für andere Werkzeuge im Entwicklungsprozess maschinenlesbar zugänglich zu machen, wird durch die MeMo-Plattform eine Erweiterung der Spezifikation von Signalschnittstellen implementiert. Die grafische Oberfläche zur Modellschnittstellenspezifikation, generiert durch die MeMo-Plattform, kann direkt in Matlab/Simulink aufgerufen werden (vgl. Abbildung 5.9).

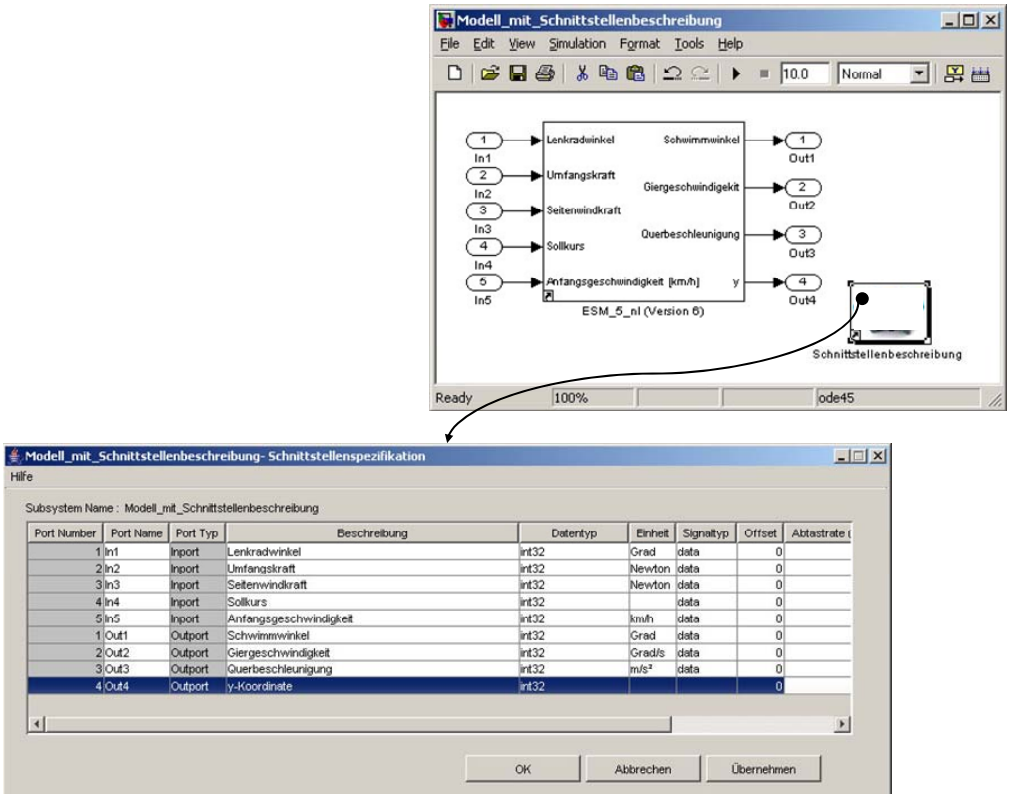


**Abbildung 5.9 - Spezifikation von Modellschnittstellen**

Vom Entwickler können zusätzlich folgende Eigenschaften zur Spezifikation der Signalschnittstellen hinterlegt werden:

- Subsystemname, Portnummer, -name und -typ
- Textuelle Beschreibung
- Datentyp
- Signaltyp
- Abtastrate

[76] In Entwicklungswerkzeugen wie beispielweise Modelica/Dymola sind die übertragenen Größen an einer Modellschnittstelle mit Masse-, Energie-, Moment-, Material- oder Informationsflüssen fest vorgegeben. [Teg05], [Mer05], [Hom06], [Ric03], [Ast98]

- Skalierung
- Offset
- Physikalische Einheit
- Minimum, Maximum

Die bereits im Modell verfügbaren Eigenschaften zu den Modellschnittstellen werden durch einen Algorithmus aus Simulink gelesen und in die Spezifikation aufgenommen. Während der Entwicklung neuer Modelle werden veränderte oder neue Schnittstellen, die noch nicht spezifiziert wurden, farbig kenntlich gemacht. Die vom Entwickler hinterlegte Schnittstellenspezifikation wird als separate XML-Datei mit dem Dateinamen des Modells im Modellierungsprojekt abgelegt. Die Strukturierung der XML-Datei basiert auf dem AUTOSAR[77]-Format, das ein Einlesen der Informationen in andere Werkzeuge während des Entwicklungsprozesses ermöglicht. Abbildung 5.10 zeigt die Struktur der XML-Datei, die im AUTOSAR-Format persistent abgelegt wird. Die gewählten Klassen innerhalb des AUTOSAR-Formats sind der Software-Komponente entnommen, ermöglichen aber neben der Beschreibung von Funktionssoftware auch die Dokumentation von Modellschnittstellen aus Streckenmodellen.

Neben der Möglichkeit, die in einem Modell vorhandenen Schnittstellen in anderen Werkzeugen einzulesen, ist ein Algorithmus zur automatischen Überprüfung der vom Entwickler verbundenen Modellschnittstellen implementiert. Beim Verbinden eines Eingangs mit dem Ausgang eines anderen Teilmodells ermöglicht ein automatisierter Vergleich der beiden XML-Dateien eine Aussage über zusammenpassende Signalflüsse. Wenn vom Entwickler beispielsweise ein Geschwindigkeitssignal mit der physikalischen Einheit $m \cdot s^{-1}$ und $km \cdot h^{-1}$ miteinander verbunden wird, folgt ein Hinweis-Fenster mit der Bitte um Überprüfung der verbundenen Modellschnittstellen. Dieser Algorithmus wird durch einen sog. Callback[78] aus Matlab/Simulink aufgerufen.

Durch die Erweiterung der MeMo-Plattform mit einem Algorithmus, der es erlaubt, innerhalb der Entwicklungsumgebung eine ergänzende Spezifikation der Modellschnittstellen vorzunehmen, wird die Basis für eine vollständige und im Modell zugängliche Dokumentation der Signalschnittstellen geschaffen. Gleichzeitig wird mit einer im Hintergrund ablaufenden Überprüfung der miteinander verbundenen Ein- und Ausgänge eine Unterstützung bereitgestellt, die das fehlerhafte Verbinden zweier Schnittstellen verhindern kann.

[77] AUTOSAR (AUTomotive Open System ARchitecture) ist ein offener Standard für Elektrik/Elektronik-Architekturen im Kraftfahrzeug. Mitglieder sind verschiedene Automobilhersteller und Zulieferer von Elektronikkomponenten [Wiki03], [Mor06].

[78] Ein Callback bezeichnet im Programmsystem Matlab einen aktionsabhängigen Funktionsaufruf, die unter gewissen Bedingungen aufgerufen wird. Simulink implementiert Callbacks, die ausgeführt werden, wenn beispielsweise ein Element kopiert, gelöscht oder geladen wird. Eine vollständige Übersicht zu Callbacks ist in [Mat05 ] zu finden.

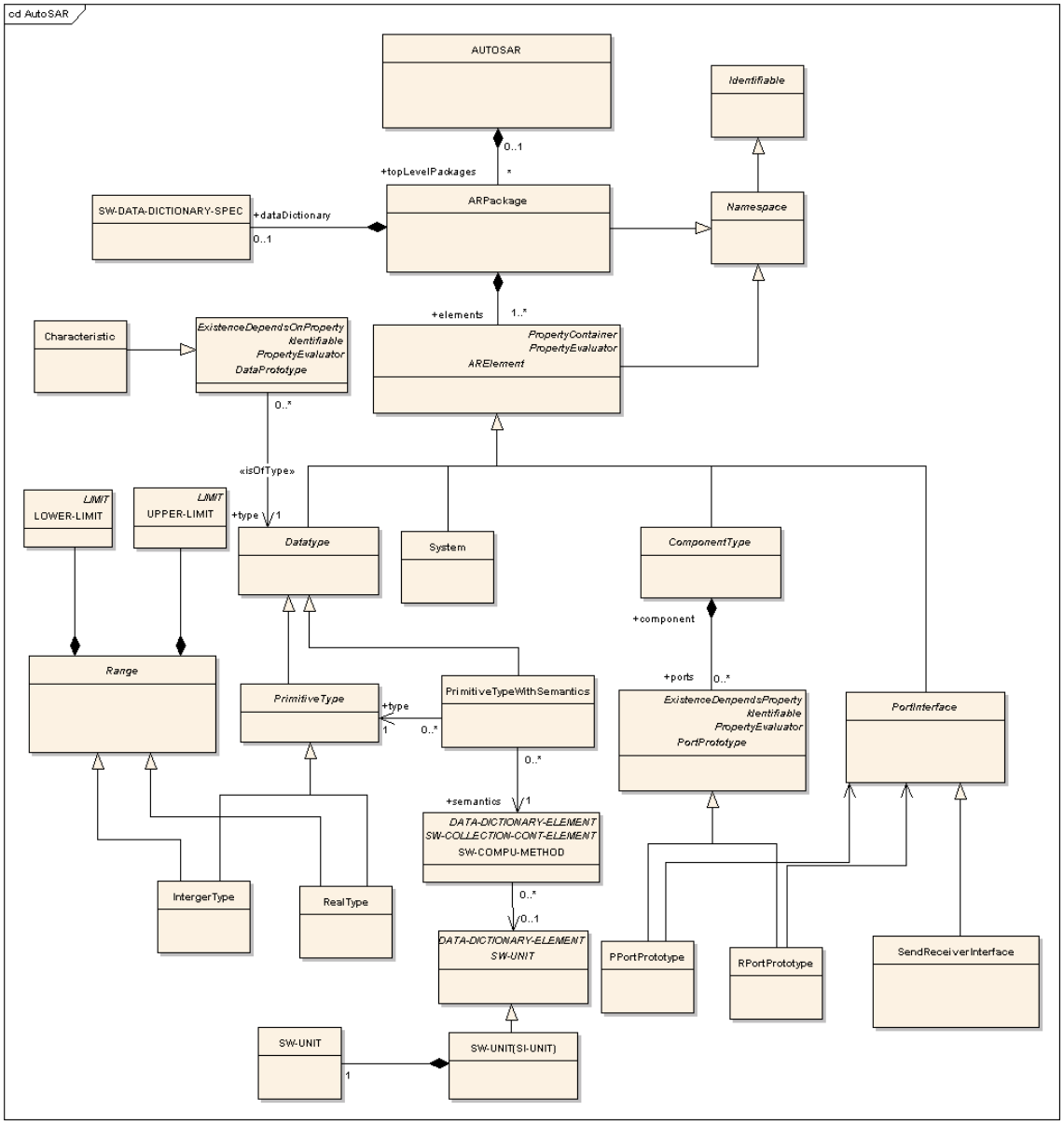


**Abbildung 5.10 - Struktur der AutoSAR-XML-Datei zur Speicherung der Signalschnittstellenspezifikation**

### 5.1.7 Abbildung des Variantenmanagements

Die in Matlab umgesetzten Signalflussdiagramme werden aus den Gleichungen des Modells abgeleitet und mit einem Parametersatz validiert. Ist die Validierung erfolgreich und somit das Modell fehlerfrei, werden Parameter und die Ergebnisse innerhalb der Dateigruppe des Modells unter Versionskontrolle gestellt. Nachdem das Modell in den Status einer Archivversion überführt wurde, kann es durch berechtigte Nutzer angewendet werden. Oftmals tritt dabei die Situation auf, dass die Parameter angepasst werden müssen, um das Modellverhalten für den spezifischen Anwendungsfall zu erhalten. Handelt es sich z. B. um ein Einspurmodell, so kann durch veränderte Parameter das dynamische Verhalten des

Modells für diverse Fahrzeuge innerhalb der Gültigkeitsgrenzen abgebildet werden. Abbildung 5.11 zeigt dies beispielhaft für drei Fahrzeuge mit unterschiedlicher Masse.

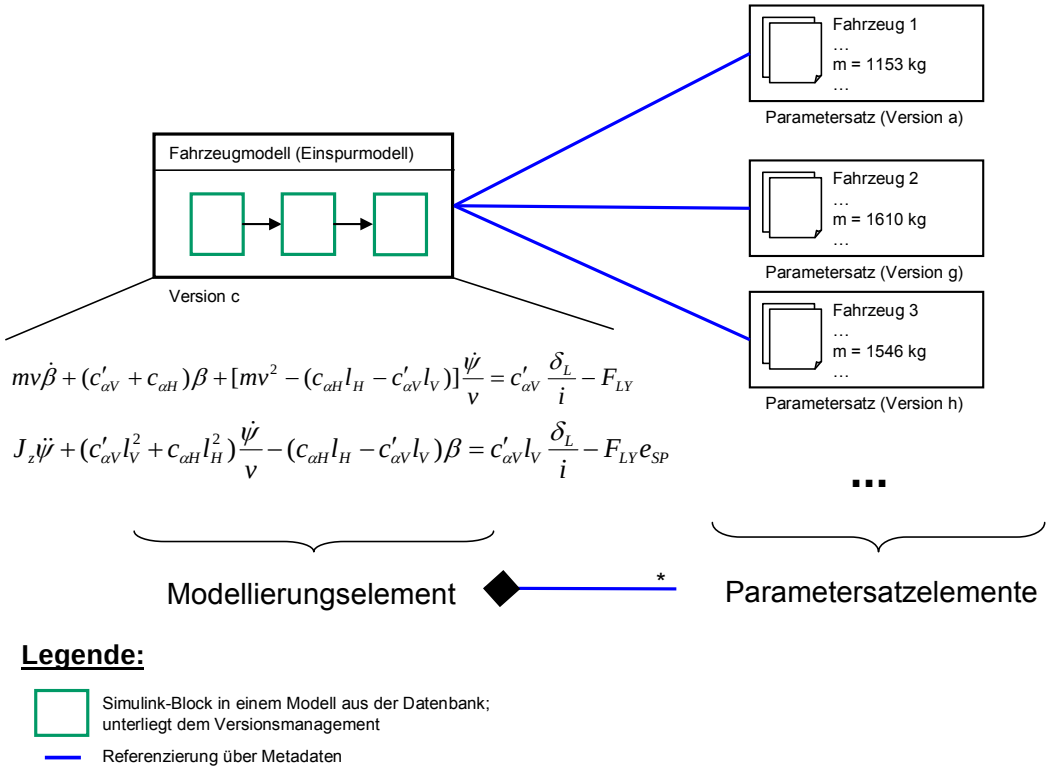


**Abbildung 5.11 - Variantenmanagement für ein Einspurmodell**

Vor dem Hintergrund der Wiederverwendung ist der Parametersatz mit Hilfe der MeMo-Plattform unter separate Versionskontrolle zu stellen. Die in der MeMo-Plattform implementierten Algorithmen bilden die Abhängigkeiten zwischen Modell und Parametersatz in dem Datenbanksystem derart ab, dass einmal erzeugte Parametersätze für alle Nutzer ersichtlich sind und sowohl Modell als auch Parametersatz separat weiterentwickelt werden können.

Die softwaretechnische Realisierung im Datenbanksystem ist so gewählt, dass kein Branch[79], wie dies in vergleichbaren Systemen der Fall ist, erzeugt wird, da dadurch für den neu erstellten Parametersatz ein Zweig in der Versionslinie des Modells entsteht, obwohl keine Veränderung des Modells vorgenommen worden ist. Für die Abbildung des Zusammenhangs zwischen Modell und Parametersatz wird von der MeMo-Plattform auf Attribute zurückgegriffen, was darüber hinaus eine Abbildung neuer Parametersätze auf festgeschriebene Archivversionen gestattet. Die zur Abbildung benötigten Attribute sind auf dem Modell- und Parametersatzelement als Attributgruppe zusammengefasst, was eine Instanziierung und somit eine Abbildung von n:m-Referenzen über die Versionshistorie der einzelnen Elemente hinweg erlaubt (vgl. Abschnitt 5.1.3). Abbildung 5.12 zeigt den Einsatz der

[79] Ein Branch stellt eine Verzweigung eines Projektes in zwei separate Versionslinien für Projekte dar.

Gruppen „Modellierungsprojekt-Referenz“ und „Parameter-Referenz“ auf den jeweiligen Elementtypen.

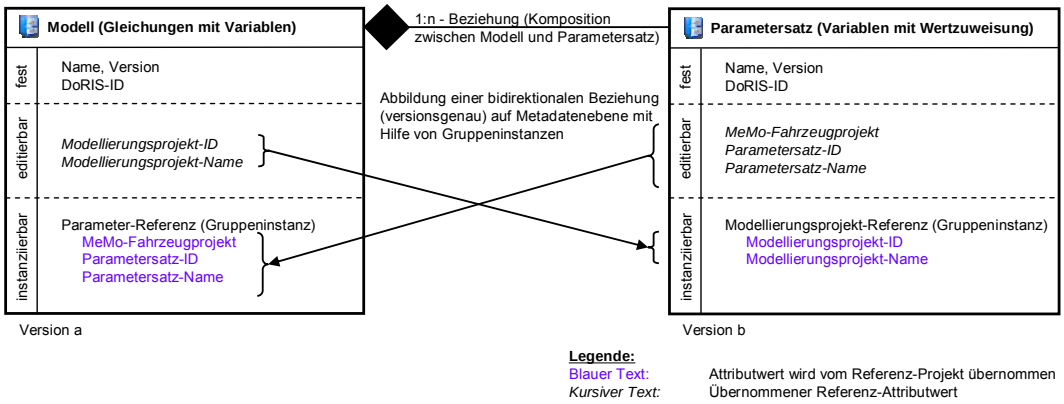


**Abbildung 5.12 - Referenzen zwischen Modell und Parametersätzen**

Gespeicherte Attributwerte beinhalten eine eindeutige Identifikation (ID) als Referenz. Die bidirektionale Abfrage nach möglichen Referenzen wird durch das Speichern der „Parametersatz-ID“ auf dem Modell und der „Modellierungsprojekt-ID“ auf dem Parametersatz ermöglicht. Wird eine Parametersatzkonfiguration auf eine Modellkonfiguration referenziert, so ist zusätzlich festzuhalten, welcher Parametersatz innerhalb der Parametersatzkonfiguration zu dem entsprechenden Teilmodell in der Modellkonfiguration zugeordnet ist. Abbildung 5.13 stellt schematisch den Zusammenhang zwischen Modell- und Parameterelementen dar. Um diese Zuordnung von Teilelementen zu gewährleisten wird die Attributgruppe „Parameterzuordnung“ (vgl. Tabelle 5.7) verwendet. Dies ist unter anderem deshalb notwendig, weil eine Parametersatzkonfiguration nicht für jedes Teilmodell einen Parametersatz beinhalten muss, sondern auch kein Parametersatz oder der „Default-Parametersatz“ innerhalb des Modells verwendet werden kann. Die Attributgruppe „Parameterzuordnung“ kann deshalb in mehreren Instanzen auf dem Element einer Parametersatzkonfiguration verwendet werden. Eine schematische Darstellung der referenzierenden Attributgruppen für Modell- und Parametersatzkonfiguration ist im Anhang unter 9.2 dargestellt.

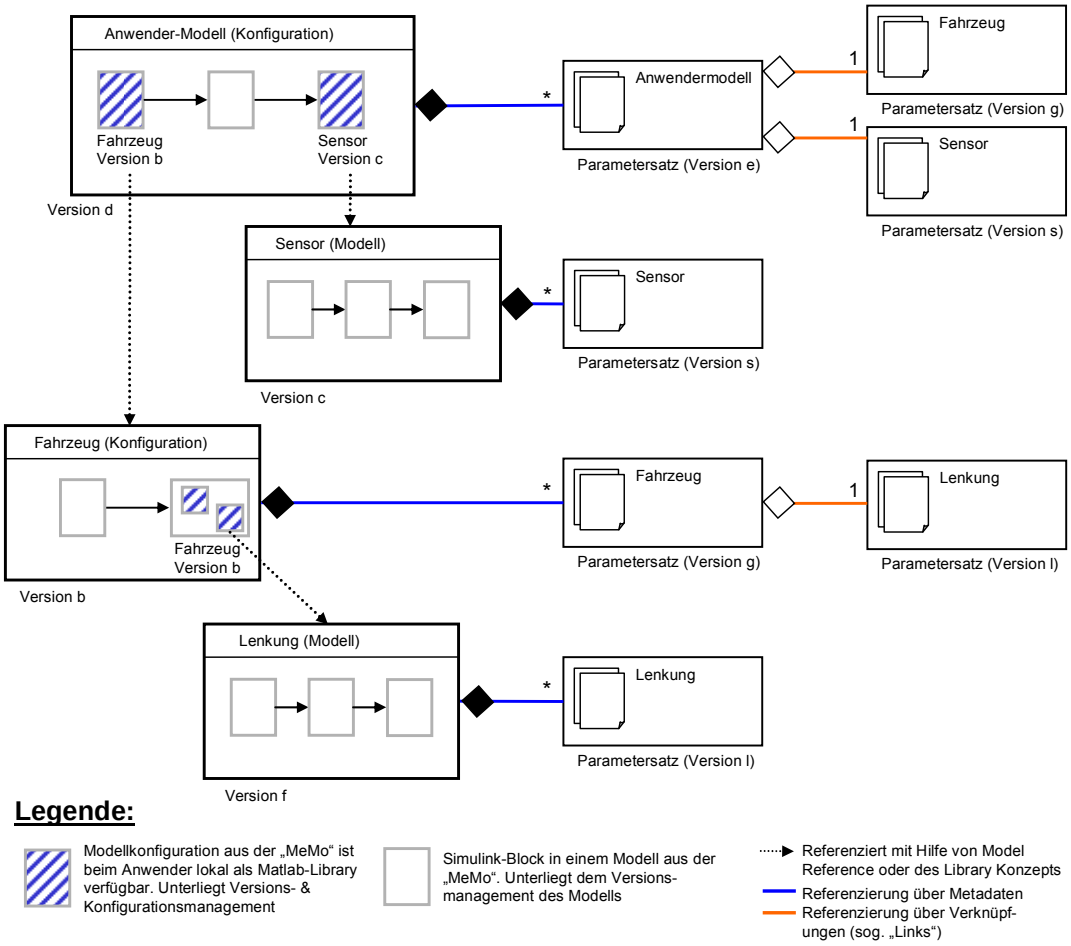


**Abbildung 5.13 - Darstellung des Konfigurationsmanagements mit Variantenmanagement am Beispiel eines Gesamtfahrzeugmodells [Bre06]**

Wird ein zusätzlich ergänzter Parametersatz durch den Nutzer für die Simulation ausgewählt, so wird durch die MeMo-Plattform vor der Simulation der „Default-Parameter“ innerhalb des Modellelements mit den Parameterdateien aus dem spezifizierten Parametersatz ausgetauscht. Die Organisation der Verzeichnispfade für das CAE-Werkzeug wird dabei ebenfalls von der MeMo-Plattform übernommen und setzt eine interaktive Kopplung zwischen Matlab und der MeMo-Plattform voraus (siehe Abschnitt 5.2).

## 5.2 Integrative Kopplung exemplarisch mit Matlab/Simulink

Unter der Zielsetzung, mit der MeMo-Plattform die Entwicklung mechatronischer Fahrzeugsysteme zu unterstützen, ist eine Middleware – mit dem Namen MeMo-Plattform – zwischen den CAE-Werkzeugen mechatronischer Fahrzeugsysteme und einem zentralen Datenbanksystem erforderlich (vgl. Kapitel 3). In dem nachstehenden Abschnitt 5.2.1 wird gezeigt, wie die Anbindung des Datenbanksystems über die MeMo-Plattform an das Entwicklungswerkzeug erfolgt. Dies wird exemplarisch an dem Werkzeug Matlab/Simulink

durchgeführt, das größtenteils von den hausinternen Anwendern eingesetzt wird (vgl. Abschnitt 4.2). Durch die Architektur der MeMo-Plattform wird sichergestellt, dass zu einem späteren Zeitpunkt noch weitere CAE-Werkzeuge integriert werden können (vgl. Abschnitt 3.2). Ziel der Kopplung zwischen Matlab/Simulink und der MeMo-Plattform ist die Erweiterung des Funktionsumfangs um die von MeMo bereitgestellten Funktionen direkt in dem CAE-Werkzeug des Anwenders. In Abschnitt 5.2.2 wird deshalb beschrieben, wie der Zugriff der in der MeMo-Plattform implementierten Algorithmen aus Matlab/Simulink erfolgen kann. Anstatt oder parallel zu Matlab/Simulink können weitere CAE-Werkzeuge wie z. B. ASCET[80] oder Dymola auf ähnliche Art und Weise an die Architektur der MeMo-Plattform angekoppelt werden.

### 5.2.1 Implementierung der Kopplung zwischen „MeMo“ und Matlab/Simulink

Das in Abschnitt 3.2 vorgestellte Makrokonzept sieht eine Fassadenklasse[81] für die Kommunikation mit der Entwicklungsumgebung vor. Diese ist in der Programmiersprache Java realisiert. Hierin sind die Methodenaufrufe für die in der MeMo-Plattform implementierten Algorithmen abstrahiert. Durch die Signatur der Methoden innerhalb der Fassadenklasse sind die Aufrufparameter für externe Programme vorgegeben. Einzelmethoden ermöglichen den Start von grafischen Benutzeroberflächen, in denen weitere Parameter zur Ausführung einer Aktion abgefragt werden. Die Realisierung einer integrativen Kopplung der Softwarewerkzeuge Matlab/Simulink und MeMo-Plattform erfordert eine bidirektionale Schnittstelle. Diese sieht vor, dass aus Matlab heraus Algorithmen der MeMo-Plattform und umgekehrt aufgerufen werden können. Es gilt aus den verfügbaren Schnittstellen der beiden Werkzeuge diejenigen auszuwählen oder ggf. weitere zu ergänzen, mit denen die Kommunikation von beiden Seiten gewährleistet werden kann. Eine Auflistung der in Matlab/Simulink für diesen Zweck vorhandenen Schnittstellen ist in Abschnitt 4.2.2 vorgenommen worden. Neben den Anforderungen bzgl. der Kommunikation ist weiterhin zu berücksichtigen, dass eine direkte Kopplung zweier Javaprogramme die gleiche Java-Runtime-Environment (JRE)[82] erfordert. Im vorliegenden Fall ist dies nicht gegeben, da die Vorgaben zur Version der JRE in Matlab und der MeMo-Plattform, die indirekt über das Datenbanksystem vorgegeben ist, divergent sind. Aus diesem Grund ist eine softwaretechnische Entkopplung der parallel laufenden JRE durch eine Socket-Verbindung[83] zu realisie-

[80] ASCET ist ein Programmsystem für die Funktions- und Softwareentwicklung von Embedded Systems in der Kfz-Elektronik.

[81] Mit einer Fassade wird einer Menge von Klassen, einem Subsystem oder einer Klassenkategorie eine einzelne einfache Schnittstelle gegeben, die von der Komplexität der Subsystem-Klassen abschirmt [Oes05].

[82] Java-Runtime-Environment (JRE) bezeichnet die Laufzeitumgebung, auf der ein Java-Programm ausgeführt wird.

[83] Als Socket bezeichnet man eine streambasierte Programmierschnittstelle zur Kommunikation zweier Rechner oder Programme über ein TCP/IP-Netz. Eine Socket-Verbindung repräsentiert somit Sockets aus der Sicht einer Client- bzw. Server-Anwendung [Kru00]. Mit der auf TCP/IP-Protokollebene durchgeführten Kommunikation wird im vorliegen Fall eine Entkopplung der zwei JREs erreicht.

ren. Weiterhin muss beachtet werden, dass das CAE-Werkzeug offiziell[84] keine Schnittstelle zur Verfügung stellt, mit der ein externes Java-Programm einen Funktionsaufruf innerhalb von Matlab/Simulink durchführen kann. Auf Grund dessen wird diese über ein Java-Native-Interface (JNI)[85], das die Kommunikation von Java auf die Programmiersprache C umsetzt, realisiert. Dies wird beispielhaft in [Web02] und [Mad05] beschrieben. Das JNI ermöglicht den Aufruf von Matlab-internen Funktionen aus Java und liefert entsprechende Rückgabewerte. Die in der exemplarisch mit dem Werkzeug Matlab aufgesetzte Architektur nutzt die in Abbildung 5.14 schematisch dargestellten Schnittstellen.

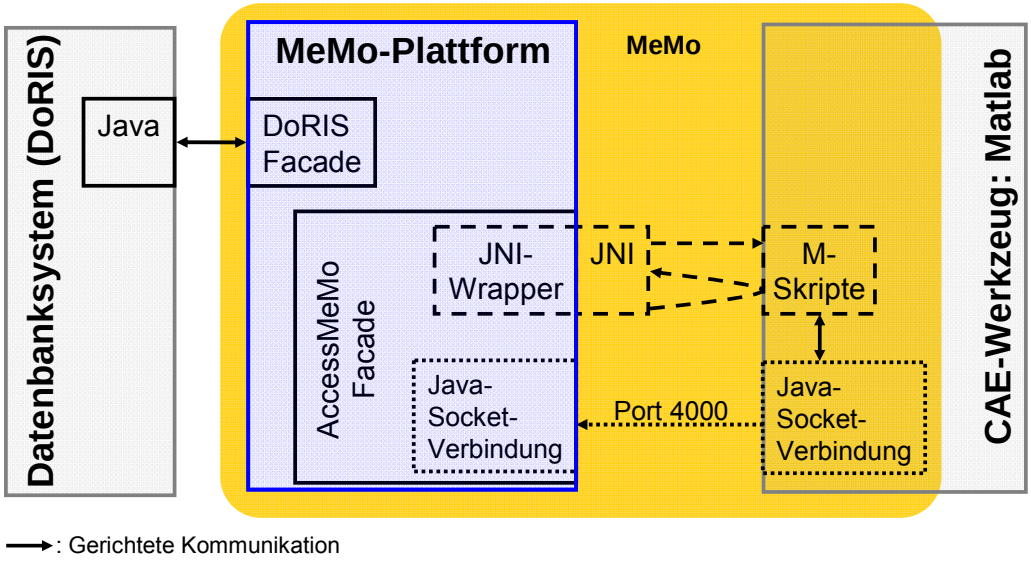


**Abbildung 5.14 - Architektur und Schnittstellen zwischen der MeMo-Plattform und Matlab**

Die Integration der MeMo-Plattform durch die oben dargestellten Schnittstellen erweitert den Funktionsumfang von Matlab um das von der MeMo-Plattform angebotene Funktionsspektrum (vgl. Abschnitt 3.3). Außerdem wird durch die von der MeMo-Plattform ergänzten Matlab-Skripte die Möglichkeit geschaffen, die MeMo-Plattform aus Matlab heraus zu steuern. Die MeMo-Plattform repräsentiert zusammen mit den Fragmenten (Java-Socket-Verbindung und M-Skripte) die Integrationsplattform MeMo. Beispiele für die aus der integrativen Kopplung resultierenden Möglichkeiten und Anwenderaktionen sowie die Beschreibung der in Matlab implementierten Fassade werden in Abschnitt 5.2.2 gegeben.

[84] Eine Realisierung kann, wie in [Mat07] oder [Web02] beschrieben, implementiert werden. Dieser Lösungsansatz wird von The Mathworks offiziell nicht unterstützt.

[85] Ein Java-Native-Interface (JNI) ermöglicht den Aufruf von nativen Funktionen C/C++ aus Java. Das JNI wird häufig aus Effizienzgründen oder zum Zugriff auf hardware- oder systemspezifische Merkmale eingesetzt [Dar02], [Ull07]. Im vorliegenden Kontext können über das JNI in Matlab definierte C-Einsprungpunkte aufgerufen werden.

### 5.2.2 Verfügbare Aktionen der „MeMo-Plattform“ in Matlab

Aus der Entwicklungsumgebung Matlab wird die Kommunikation mit der MeMo-Plattform immer über den Java-Bestandteil durchgeführt, der von der MeMo-Plattform mit in den Matlab-Kern eingebunden wurde. Die Kommunikation mit der MeMo-Plattform ist über die Socket-Verbindung (vgl. Abschnitt 5.2.1) umgesetzt. Aufgerufen werden kann jede öffentliche Java-Methode in der Fassadenklasse der MeMo-Plattform. Die in Matlab enthaltene Methode `javaMethod` ermöglicht dabei den Zugriff, der z. B. für das Herunterladen eines Projektes aus dem Datenbanksystem durch folgende Matlab-Befehlsfolge aufgerufen wird:

```
% Erstellen einer Variable zur Übergabe der Parameter
%  für die Methode "downloadProjekt"
parameter = javaArray('java.lang.String', 4);
parameter(1) = java.lang.String(ExternalLink);
parameter(2) = java.lang.String(LokalerPfad);
parameter(3) = java.lang.String(ZielBlock);
parameter(4) = java.lang.String(ZielSystem);
% Aufruf der Methode "downloadProjekt" in der
%  Fassadenklasse der MeMo-Plattform
returnValue = javaMethod ('downloadProject',...
de.vwag.modellbibliothek.memoplattform.AccessMeMoFacade, parameter);
```

Durch ein Matlab-Skript wird dem Anwender eine komfortable Fassade für die verfügbaren Methoden der MeMo-Plattform-Fassade angeboten. Neben einer gekapselten Schnittstelle zu der MeMo-Plattform kann der Aufruf einer Methode auf eine Zeile reduziert werden. Die über MeMo verfügbaren Skripte stellen in Matlab die Methoden in Tabelle 5.9 für die interaktive Anwendung oder eine zukünftige Erweiterung (vgl. Kapitel 7) bereit.

| Methodenname der Matlab-Fassade (MeMo_DBfkt.m) | Ausgeführte Aktion in der MeMo-Plattform |
|---|---|
| DBinit | Aufruf des Datenbank-Browsers als Simulink-Bibliothek. |
| MeMoFokus | Setzt zu dem in Matlab selektierten Block den entsprechenden Fokus in der MeMo-Plattform. |
| downloadProjekt | Herunterladen eines Elements aus dem Datenbanksystem und Bereitstellung der Pfade in der Entwicklungsumgebung. |
| downloadDoku | Herunterladen der Dokumentation zu einem Modell und Öffnen in dem geeigneten Viewer. |
| allVersions | Zeigt alle verfügbaren Versionen zu einem Element an. |
| searchParams | Recherche nach im Datenbanksystem verfügbaren Parametersätzen zu einem Modell. |
| getInfo | Abrufen der Metadaten zu einem Element. |
| setParameterset | Parametrisieren eines Modells mit einem separaten Parametersatz. |

| changeProjectVersion | Wechseln der lokal vorhanden Projekt-Version mit einer anderen Version im Datenbanksystem. |
|---|---|
| checkoutElement | Auschecken eines Elements. |
| checkinElement | Einchecken eines Elements. |
| archivElement | Archivieren eines Elements. |

**Tabelle 5.9 - Aus der Fassade der MeMo-Plattform bereitgestellte Methoden für Matlab**

Neben den Aufrufparametern für die Methoden sind teilweise weitere Parameter für die durchgeführten Aktionen notwendig. Das Einchecken oder Archivieren von Elementen erfordert beispielsweise die Spezifikation der Metadaten. Zu diesem Zweck werden die grafischen Oberflächen von der MeMo-Plattform eingeblendet. Um für den Nutzer einen möglichst komfortablen Zugriff auf die Modelle zu gewährleisten sind im Funktionsumfang von MeMo Methoden implementiert, die den Inhalt des Datenbanksystems in Form einer Simulink-Bibliothek aufbereiten. Wie in Abbildung 5.15 dargestellt, werden die verfügbaren Modelle sowie weitere Buttons zum Abrufen archivierter Versionen und der Metadaten bzw. weiterer Dokumentationen angeboten.

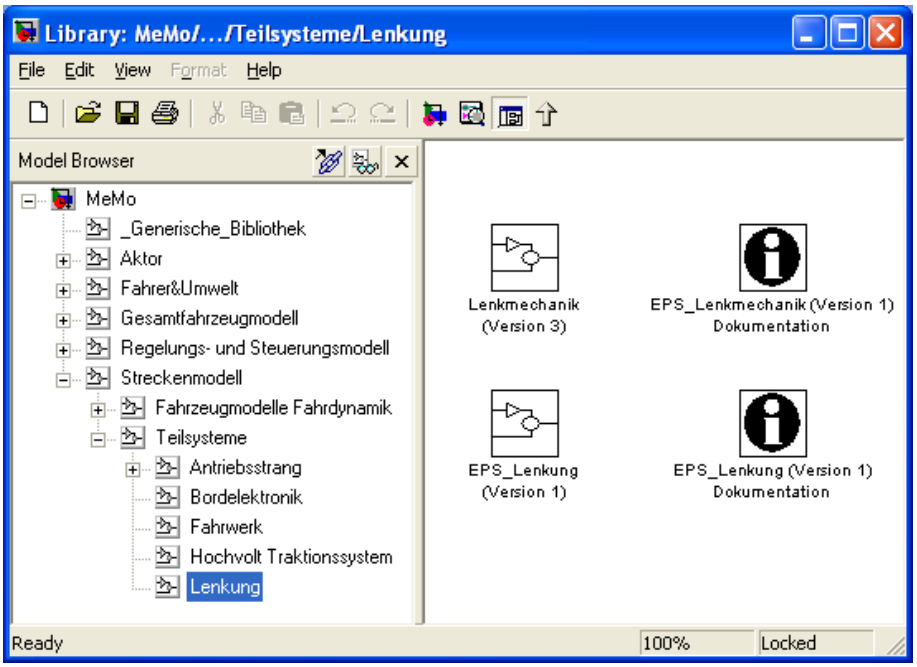


**Abbildung 5.15 - Sicht auf die Daten im Datenbanksystem in Form einer Simulink-Bibliothek [Bre07]**

Die Bibliothek stellt den Anwendern die Modelle für eine flexibel konfigurierbare Modellumgebung in Matlab/Simulink zur Verfügung. Die Elemente können daraus wie vom Nutzer gewohnt per Drag&Drop entnommen werden. Während dem Drag&Drop-Vorgang erfolgt im Hintergrund durch die Algorithmen der MeMo-Plattform das Herunterladen des Modells und das Bereitstellen der Pfade in dem CAE-Werkzeug. Über zusätzlich ergänzte Buttons auf der Matlab-Oberfläche sind weitere Aktionen über die grafische Interaktion des

Nutzers zu starten. Sobald ein Modell, das aus dem Datenbanksystem stammt, selektiert wird, sind folgende Aktionen wählbar:

- Modell parametrisieren
- Version eines Modells wechseln
- Fokus des selektierten Modells auf der MeMo-Plattform setzen
- Simulink-Bibliothek mit den Elementen im Datenbanksystem öffnen

Durch die Integration der MeMo-Plattform in die Entwicklungsumgebung Matlab entsteht keine Einschränkung der in Matlab/Simulink standardmäßig vorhandenen Funktionalitäten. Vielmehr werden durch die Integration des zusätzlichen Funktionsumfangs in Matlab/Simulink dem Nutzer neue Möglichkeiten für die täglich benötigten Aufgaben bereitgestellt. Dies ermöglicht dem Entwickler eine Nutzung vorhandener Modelle bzw. eine Erstellung neuer Modelle mit einem ihm gewohnten Werkzeug durchzuführen.

# 6 Bereitstellung von „MeMo“ und Applikation am Beispiel

Nach der Erläuterung des Konzeptes für die MeMo-Plattform (vgl. Abschnitt 3.2) und des darin geplanten Funktionsspektrums (vgl. Abschnitt 3.3) wurden ausgewählte Algorithmen (vgl. Kapitel 5) detaillierter vorgestellt. Als erstes CAE-Werkzeug für die Implementierung wurde Matlab/Simulink ausgewählt, das, wie in Abschnitt 5.2 gezeigt, mit der MeMo-Plattform verkoppelt wird. In diesem Kapitel wird kurz auf die Implementierung der MeMo-Plattform eingegangen und beschrieben, wie diese dem Nutzer bereitgestellt wird. Die Unterstützung während der Modellbildung oder der Funktionsentwicklung kann sowohl in Matlab/Simulink als auch auf der MeMo-Plattform direkt erfolgen. In jedem Fall stellt jedoch die MeMo-Plattform die Grundlage für die Kommunikation des lokalen Clienten mit dem Datenbanksystem dar. Selbst eine aus Matlab gestartete Aktion, wie beispielsweise das Anzeigen zusätzlicher Informationen zu einem Element, bedient sich an dem zur Verfügung stehendem Funktionsumfang der MeMo-Plattform. Im den nachfolgenden Abschnitten soll aus diesem Grund vorerst auf die Bereitstellung der MeMo-Plattform für den Nutzer eingegangen werden, bevor der Umgang mit Matlab/Simulink dargestellt wird. Die in Abschnitt 6.4 gezeigte Applikation stellt einen Teil des kompletten Funktionsumfangs am Beispiel eines Fahrzeugmodells mit zwei Regelsystemen dar.

## 6.1 Bereitstellung für den Nutzer

Das in Abschnitt 3.2 vorgestellte Makrokonzept der MeMo-Plattform ist in der Programmiersprache Java umgesetzt. Dies ermöglicht den plattformunabhängigen Einsatz der MeMo-Plattform und unterstützt die Java-basierte Kommunikationsschnittstelle der meisten Versionskontrollsysteme. Für die Implementierung wird auf eine vielfach in der Softwaretechnik bewährte Plattform zurückgegriffen. Das über das Open Source Projekt „Eclipse“ verfügbare Gerüst der „Rich Client Platform“ (RCP)[86] wird für die MeMo-Plattform verwendet. Die RCP enthält bereits eine Basisausstattung für die Entwicklung von Softwareapplikationen, wie beispielsweise SWT/JFace[87] oder das OSGi-Framework[88]. Es ist durch verschiedene Module und Plug-Ins erweiterbar, was bei vielen Problemstellungen die Ver-

[86] Die Rich-Client-Plattform bezeichnet ein Java-Framework, das durch Module oder Plug-Ins erweiterbar ist. Komplexe, vorgefertigte Algorithmen, beispielsweise zur Darstellung einer Hilfefunktion, Installation oder Aktualisierung können als separates Modul eingebunden und konfiguriert werden [Mca05].

[87] SWT steht für „Standard Widget Toolkit“ und bezeichnet eine Bibliothek für die Erstellung grafischer Oberflächen in der Programmiersprache Java. SWT nutzt die nativen grafischen Elemente des Betriebssystems. JFace basiert auf den SWT Basiskomponenten und stellt vorgefertigte komplexere Benutzeroberflächen für Java bereit [Hat04], [Har04].

[88] Das „Open Services Gateway initiative“ (OSGi) ermöglicht die Organisation und Steuerung des Lebenszyklus der Plugins und bildet damit das Fundament der Eclipse RCP. Plugins können während der Laufzeit installiert, ausgeschaltet oder ausgetauscht werden. Weiterhin werden dabei die Abhängigkeiten zwischen den einzelnen Plugins verwaltet [Arn07].

wendung vorgefertigter Komponenten erlaubt. Im vorliegenden Kontext werden die Folgenden für die Implementierung von Standard-Funktionalitäten eingesetzt:

- Update-Manager (Stellt ein Modul bereit, durch das in einer Client-Server-Architektur eine Aktualisierung der Client Software ermöglicht wird.)
- Hilfe-Plugin (Stellt ein komfortables Modul für die Hilfe bzw. Dokumentation eines Programmsystems bereit.)

Die daraus entstandene Applikation mit der grafischen Oberfläche, wie in Abbildung 6.1 gezeigt, ermöglicht dem Nutzer das Arbeiten mit der MeMo-Plattform.

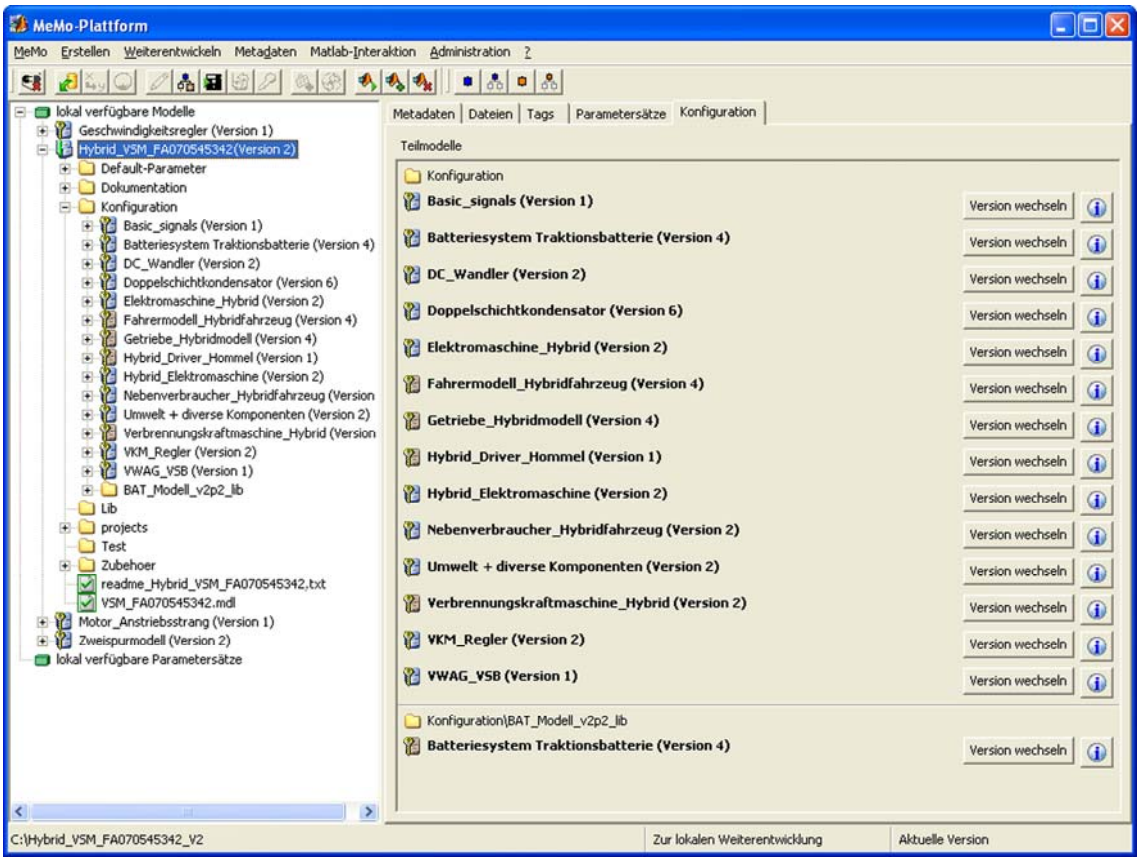


**Abbildung 6.1 - Grafische Oberfläche der MeMo-Plattform**

Die Installation der MeMo-Plattform erfolgt auf der lokalen Festplatte des Nutzers. Im Anschluss wird vom Nutzer die Konfiguration der MeMo-Plattform vorgenommen. Unter anderem können Parameter, wie der Einstiegsknoten im Datenbanksystem, persönliche Einstellungen und die Spezifikation des CAE-Werkzeugs vorgenommen werden. Nach der Auswahl der entsprechenden Matlab-Version werden von der MeMo-Plattform Dateien im Installationsverzeichnis von Matlab ergänzt und diese über die Pfadvariablen in den Matlab-Kernel eingebunden. So wird letztendlich garantiert, dass die MeMo-Plattform mit der richtigen Matlab-Version auf dem Computer des Nutzers interagiert. Ein nachträgliches Ändern der Konfiguration ist problemlos durchführbar. Die Interaktion kann ebenso deaktiviert und die MeMo-Plattform als selbstständig lauffähige Applikation betrieben werden.

Basierend auf aktivierten Interaktionen zwischen der MeMo-Plattform und dem Entwicklungswerkzeug ist für den Nutzer ebenso ein ausschließliches Arbeiten in Matlab/Simulink möglich. Die im Datenbanksystem vorhandenen Elemente können über eine Simulink-Bibliothek dargestellt und verwendet werden (vgl. Abschnitt 5.2.2). Während dem Entwicklungsprozess sind Aktionen wie Auschecken, Einchecken oder Archivieren durch zusätzlich in Matlab ergänzte Schaltflächen oder über die Matlab-Kommandozeile möglich. Nach dem Aufruf der Aktion werden nach Bedarf grafische Oberflächen der MeMo-Plattform angezeigt, um benötigte Informationen bzw. Metadaten zu komplettieren.
Eine Auswahl des Funktionsspektrums über das CAE-Werkzeug Matlab/Simulink bzw. die MeMo-Plattform wird in Abschnitt 6.2 beschrieben.

## 6.2 Vorgehensweise während der Entwicklung

Vorerst soll ein denkbarer Arbeitsablauf gezeigt werden. Unterschieden wird in der Vorgehensweise zwischen dem generischen und dem mechatronischen Modell. Basierend auf diesen werden Teilaspekte des kompletten Funktionsspektrums gesondert betrachtet und jeweils erläutert. Es wird dargestellt, in welcher Form der Anwender von MeMo während der Modellbildung und Funktionsabsicherung von Steuergerätefunktionen profitiert. Fortführend wird ein Querschnitt der Funktionalitäten vorgestellt, die für den Anwender direkt in seinem gewohnten Entwicklungswerkzeug zur Verfügung stehen und darin genutzt werden können.

**Generische Modelle und mechatronische Fahrzeugsysteme**

Wie im Abschnitt 3.1 gezeigt, kann für die Entwicklung bzw. Anwendung von Modellen zwischen einem generischen und einem mechatronischen Modell unterschieden werden. Für beide Modellgruppen ist in der Abbildung 6.2 und Abbildung 6.3 ein denkbares Ablaufdiagramm für die Entwicklung dargestellt. Die Vorgehensweise mit dem generischen Modell stellt einen Ausschnitt der Vorgehensweise innerhalb des mechatronischen Modells dar. Die Anwendung eines mechatronischen Modells oder einer mechatronischen Komponente kann analog zum dargestellten Vorgehen eines generischen Modells stattfinden. In den Diagrammen wird bzgl. des Einstiegspunktes nicht zwischen den unterschiedlichen Rollen: Anwender, Entwickler und Projektleiter unterschieden.

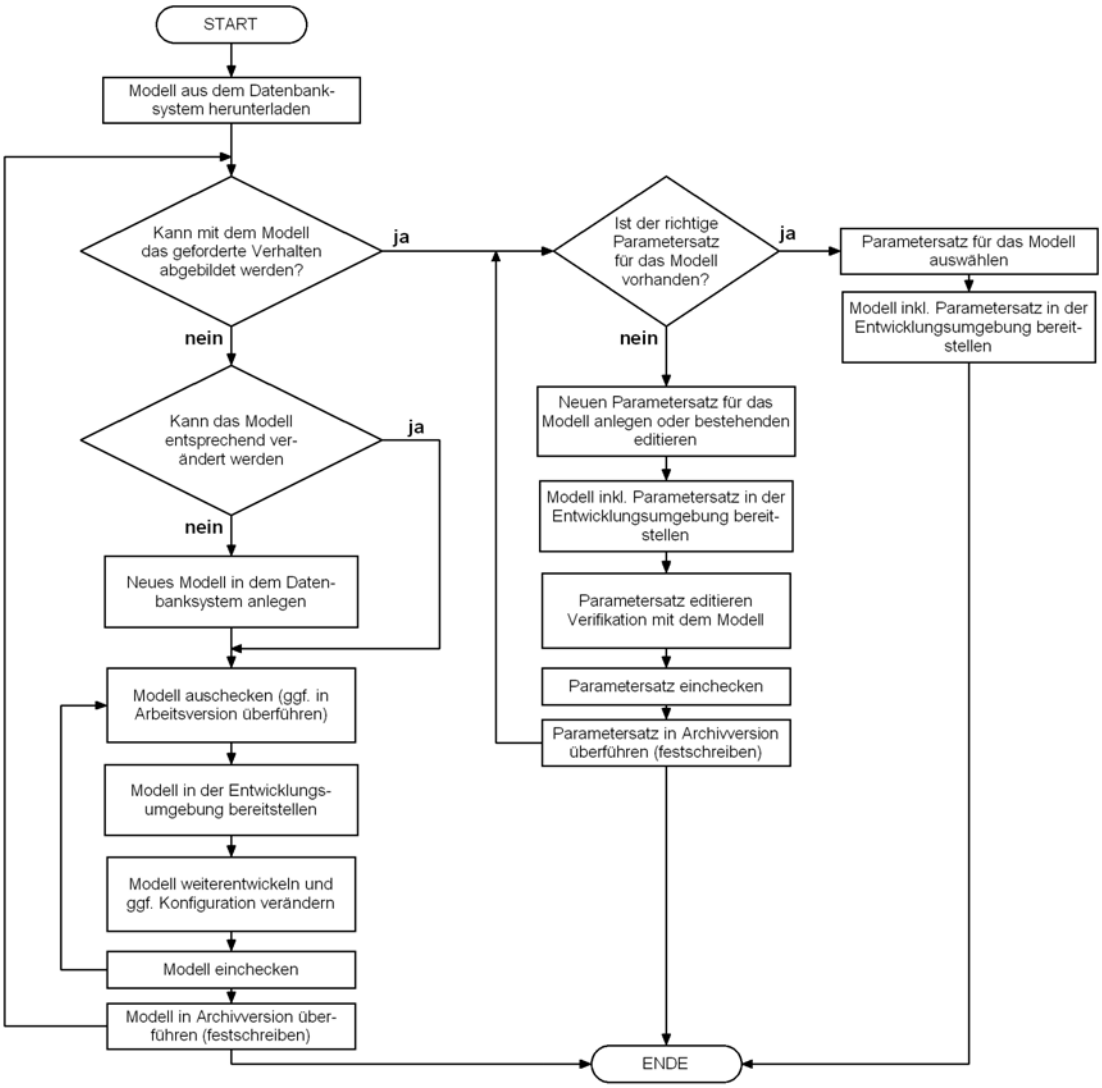


**Abbildung 6.2 - Ablaufdiagramm generisches Modell**

Das Ablaufdiagramm für die Anwendung bzw. die Entwicklung des generischen Modells beginnt mit dem Lesen der Metadaten sowie dem Herunterladen eines Modells auf die lokale Festplatte des Computers des Anwenders. Basierend auf der Modellprogrammierung kann eine Entscheidung für den Einsatz zur Simulation getroffen werden. Ist die Modellierungstiefe für das vom Anwender gewünschte Verhalten ausreichend, wird im nächsten Schritt nach Parametern recherchiert. Anderenfalls kann das Modell bearbeitet werden bzw. ein neues Modell erstellt werden, das in der gewünschten Modellierungstiefe implementiert wird. Nach erfolgreicher Parametrisierung bzw. Modellierung kann die Simulation in dem CAE-Werkzeug erfolgen. Die dargestellte Vorgehensweise ist ein Bestandteil der Arbeitsweise jedes einzelnen Entwicklers während eines durch mehrere Entwickler parallel durchgeführten Entwicklungsprozesses an einem mechatronischen Fahrzeugsystem. Ebenso kann die Vorgehensweise für eine mechatronische Komponente in einem mechatronischen Gesamtsystem eingesetzt werden (vgl. Abschnitt 2.1.2). Das Ablaufdiagramm für mechatronische Fahrzeugsysteme ist in Abbildung 6.3 dargestellt.

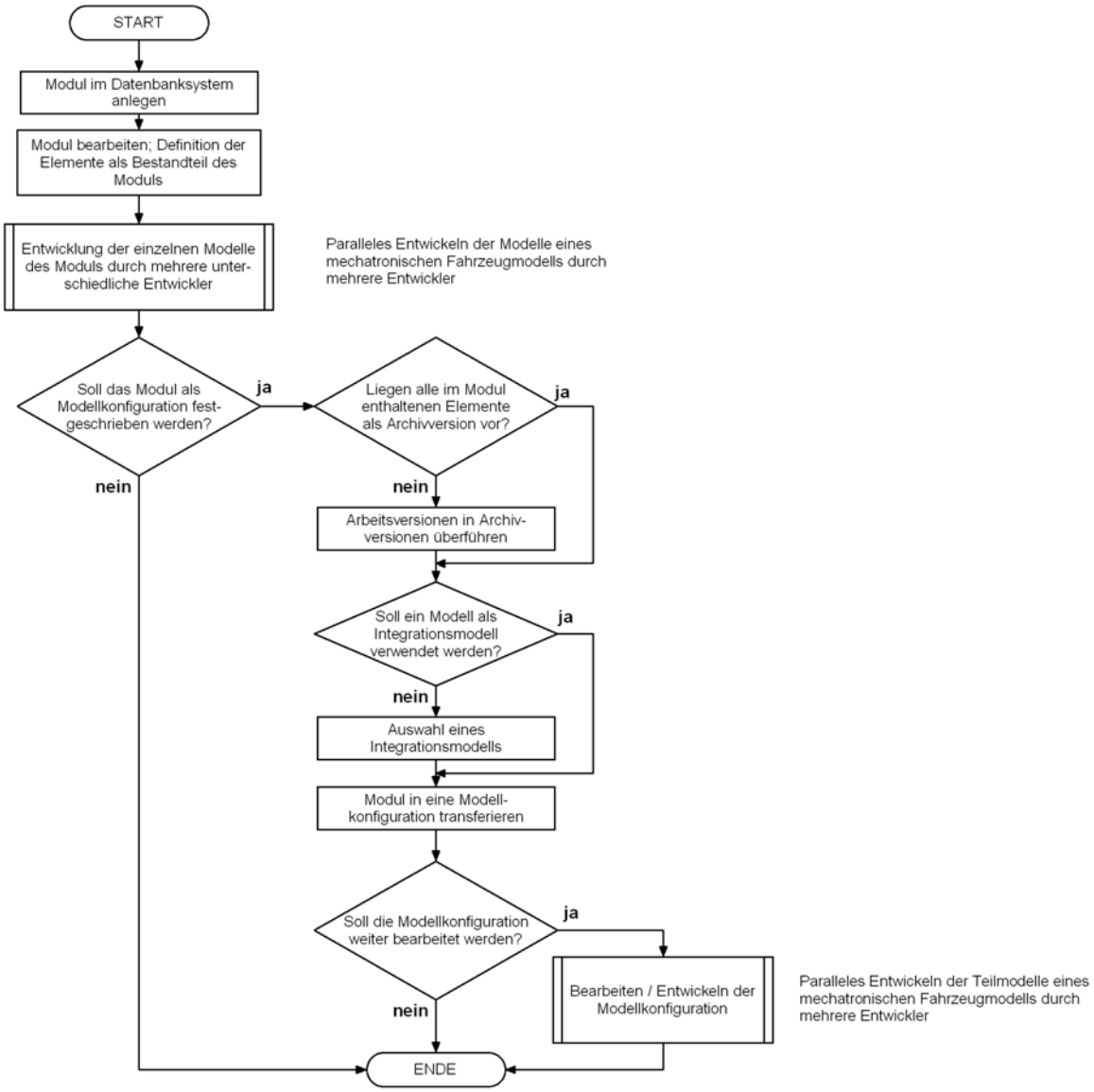


**Abbildung 6.3 - Ablaufdiagramm für das mechatronische Modell**

Das Ablaufdiagramm für mechatronische Modelle ist verdichtet dargestellt, dies bedeutet, dass die Entwicklung einzelner Modelle oder Module nach demselben Schema wie das Entwickeln eines generischen Modells erfolgt. Die Besonderheit im Kontext mechatronischer Fahrzeugsysteme ist das parallele Arbeiten an verschiedenen Modellen innerhalb eines Moduls oder einer Modellkonfiguration durch mehrere Entwickler (vgl. Abschnitt 5.1.5). Das Modul kann in eine Modellkonfiguration überführt werden.

## 6.3 Paralleles Arbeiten durch mehrere Entwickler an einem Modell

Wie in Abschnitt 6.2 gezeigt, ist das parallele Arbeiten durch Entwickler unterschiedlicher Domänen sowohl an Modulen als auch an Konfigurationen möglich. Dies wird in den Abschnitten 6.3.1, 6.3.2 und 6.3.4 nun im Detail betrachtet.

### 6.3.1 Dynamische Änderungen an gruppierten Elementen ohne festgehaltene Version

Die stetige Weiterentwicklung von Teilmodellen durch unterschiedliche Entwickler, die an der Entwicklung eines mechatronischen Gesamtmodells beteiligt sind und die daraus entstehende Forderung (vgl. Abschnitt 3.1.2), alle Teilmodelle bei den beteiligen Entwicklern aktuell zu halten, wird durch das Modul optimal abgedeckt. Das Modul gruppiert einzelne Modelle, ohne explizit die Version festzuhalten. Die Versionskontrolle für Dateien und komplette Modelle erfolgt nach dem im Abschnitt 5.1.2 dargestellten Schema. Das Herunterladen bzw. Aktualisieren eines Moduls auf dem Computer des Entwicklers ermöglicht, wie in Abbildung 5.4 dargestellt, eine freie Auswahl der Version darin enthaltener Elemente. Liegt auf dem Computer des Entwicklers ein Element vor, das in dem Datenbanksystem in einer aktuelleren Version verfügbar ist, so wird dies textuell und durch ein farblich gekennzeichnetes Symbol an dem Modell- oder Parametersatzelement dargestellt (vgl. Abbildung 6.1). Der Entwickler erkennt dadurch sofort, wenn ein anderer Entwickler Modelle, die im Modul gruppiert sind, aktualisiert hat. Durch ein explizites Wechseln der Version eines einzelnen Elementes kann das lokal verfügbare Modul schrittweise auf den aktuellen Entwicklungsstand gebracht werden. Ebenso sind Operationen auf das gesamte Modul möglich, was ein erneutes Herunterladen aller neu verfügbaren Versionen auf den Computer des Entwicklers erlaubt.
Während der Entwicklung mechatronischer Fahrzeugsysteme findet das Modul insbesondere in Projekten Einsatz, die sehr dynamisch durch mehrere Entwickler parallel entwickelt werden. Die innerhalb eines Moduls gruppierten Modelle können von jedem Entwickler mit Ändern-Recht auf das Modul editiert werden. Ein Entwicklungsstand kann über alle im Modul gruppierten Elemente mit Hilfe eines Tags dokumentiert werden. Dieser kann auf beliebige Versionen ergänzt und später wieder entfernt werden. Dieses ermöglicht keine Reproduktion eines Moduls. Aus diesem Grund kann ein Modul inklusive der darin enthaltenen Elemente in einer Modellkonfiguration festgeschrieben werden. Hierzu wird, wie in Abbildung 6.3 gezeigt, ein Integrationsmodell innerhalb des Moduls ausgewählt, in das die verbleibenden Elemente in einer vom Projektleiter spezifizierten Version abgelegt werden.

### 6.3.2 Sporadische Änderungen an festgeschriebenen Elementen

Ähnlich wie das Modul gruppiert auch die Modellkonfiguration unterschiedliche Teilelemente. Im Gegensatz zum Modul enthält die Konfiguration selbst Dateien und Verzeichnisse und ermöglicht eine hierarchische Gliederung weiterer Modellkonfigurationen oder Modellierungsobjekte. Die hierarchische Gliederung von Modellen und Modellkonfigurationen ermöglicht den Aufbau komplexer Simulationsmodelle aus unterschiedlichen Domänen, in denen jedes Teilelement versionsgenau festgehalten wird. Die Abbildung der Referenzen innerhalb der Modellkonfiguration stellt eine Wiederverwendung und die Reproduktion in sich abgeschlossener Modellbestandteile sicher. Jedes Element inkl. aller zugehörigen Dateien unterliegt der Versionskontrolle (vgl. Abschnitt 5.1.2). Die Modellkonfiguration enthält typischerweise das Integrationsmodell, in dem die hierarchisch auf erster Ebene gegliederten Elemente in der Entwicklungsumgebung zusammengeführt werden. Die Ver-

wendung der in Matlab/Simulink bereitgestellten Konzepte zur Organisation von hierarchisch gegliederten Modellen setzt die Verfügbarkeit aller Modell-Bibliotheken und -Referenzen im Suchpfad des Entwicklungswerkzeugs Matlab voraus. Die MeMo-Plattform stellt sicher, dass alle Verzeichnispfade, die für die Simulation benötigt werden, in dem Entwicklungswerkzeug verfügbar sind. Einsatz findet die Modellkonfiguration überwiegend in Projekten, in denen enthaltene Teilelemente sporadisch und ggf. parallel weiterentwickelt werden. Ebenso wie für die Teilelemente innerhalb eines Moduls wird auch hier über den Status kenntlich gemacht, sobald neue Versionen von Teilelementen im Datenbanksystem verfügbar sind. Eine Aktualisierung von Teilelementen oder ein Versionswechsel, wie beispielsweise beim Integrationstest erforderlich, bedingt aufgrund der Reproduzierbarkeit ein neues Versionieren aller hierarchisch übergeordneten Elemente. Durch diesen Mechanismus kann ebenfalls sichergestellt werden, dass Parametervarianten inklusive der ggf. enthaltenen Parametersatzobjekte entsprechenden Modellelementen zugeordnet werden können (vgl. Abschnitt 5.1.7).

### 6.3.3 Datenvarianten auf Modelle einer Konfiguration

Die Auswahl der Parametersatzelemente für die Modellkonfiguration wird durch die MeMo-Plattform dadurch unterstützt, dass der Entwickler die im Datenbanksystem vorhandenen Elemente über Auswahllisten zur Verfügung gestellt bekommt, wie in Abbildung 6.4 dargestellt. Im Anschluss an die Spezifikation der Parametersatzelemente für die in der Modellkonfiguration enthaltenen Elemente erfolgt das Herunterladen auf den Computer des Entwicklers.

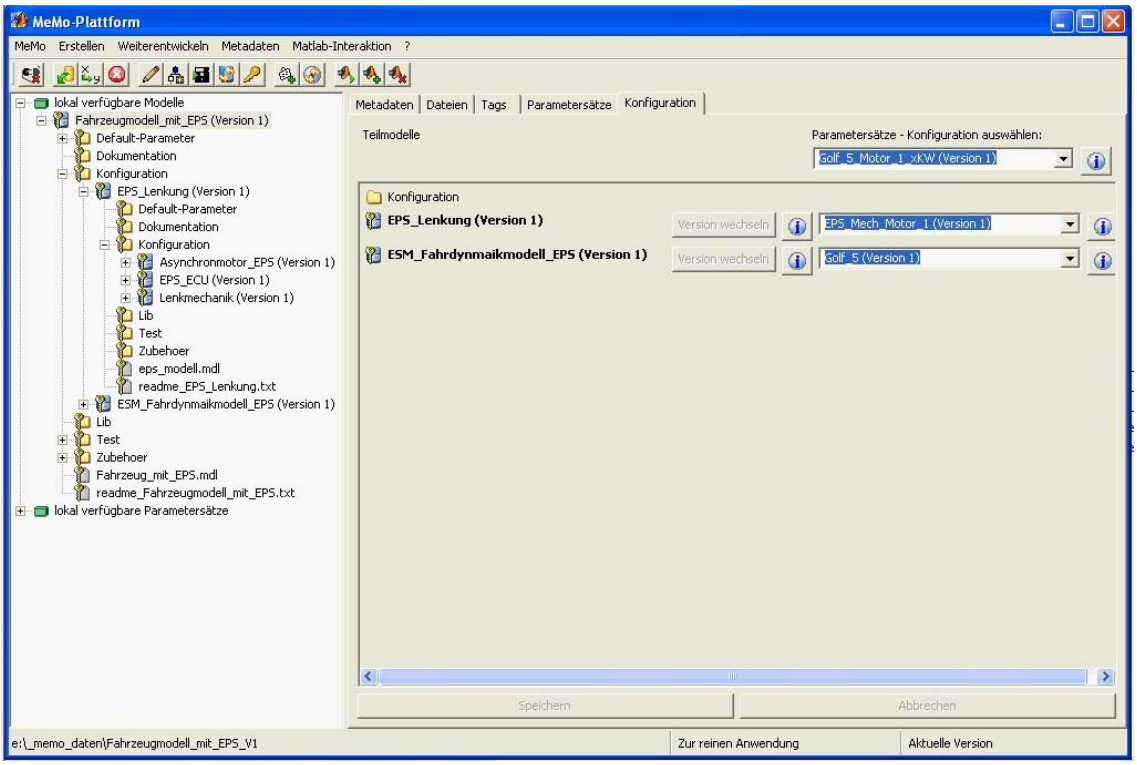


**Abbildung 6.4 - Auswahl der Parametersatzelemente für eine Konfiguration**

Die ausgewählten Parametersätze werden für die Simulation in der Entwicklungsumgebung zur Verfügung gestellt, anstelle der in den Modellierungselementen vorhandenen „Default-Parameter“. Auf eine Modellkonfiguration lassen sich beliebig viele Parametersatzelemente anlegen, die für die Simulation unterschiedlicher Varianten sukzessive in Matlab zur Verfügung gestellt werden. Seitens Matlab erfolgt mit dem Laden der Parametersätze für das Modell eine Parameteranalyse. Doppelt vorhandene Parameter mit unterschiedlichen Werten werden erkannt und der Entwickler darauf hingewiesen. Insbesondere nach dem Aufbau einer neuen Konfiguration aus unterschiedlichen Teilelementen hilft diese Funktion mögliche Fehler durch die Parametrisierung des Modells zu verhindern.

### 6.3.4 Arbeiten in Form von Softwarekooperationen

Immer mehr neue Funktionen, durch die die Steuergeräte untereinander noch stärker miteinander vernetzt werden, halten Einzug in die Fahrzeugelektronik und fordern zugleich einen intensiveren Austausch der Modellierer. Die geforderte interdisziplinäre Kooperation zwischen den Entwicklern mechatronischer Systeme ist eine der wesentlichen Forderungen für das synchrone Zusammenarbeiten von Entwicklergruppen. Das Arbeiten in einem Team von Entwicklern ist typisch für den integrativen Entwicklungsprozess von mechatronischen Systemen, da unterschiedliche Disziplinen durch verschiedene Organisationseinheiten und räumlich getrennte Entwickler abgedeckt werden. Ermöglicht werden muss die Optimierung mechatronischer Systeme im Systemverbund der am mechatronischen System beteiligten Disziplinen. Dies setzt einen Prozess mit transparenten Frei- und Weitergaben von Teilmodellen (vgl. Abschnitt 5.1.2) sowie die Möglichkeit der parallelen Entwicklung an komplexen Modellen voraus. Das parallele Arbeiten an einem Modell als Modul oder Modellkonfiguration bedingt eine separate Versionierung der modularen Teilmodelle und deren Zusammenführung in einer übergeordneten Instanz, wie dies durch MeMo implementiert ist (siehe Abschnitte 6.3.1 und 6.3.2). Die Zusammenarbeit kann in Form von Softwarekooperationen (vgl. Abbildung 5.7) organisiert werden. Dabei wird unterschieden zwischen der

- Zusammenarbeit unternehmensinterner Entwickler mit Lieferanten und der
- Zusammenarbeit unternehmensinterner Entwickler, übergreifend zwischen Organisationseinheiten.

Die Zusammenarbeit mit externen Lieferanten erfolgt für einen Entwicklungsauftrag über das Webportal, wie in Abbildung 5.8 dargestellt. Vorraussetzung hierfür ist, dass zu definierten Zeitpunkten (z. B. zum Projektstart und -ende) ein Austausch der Modelle erfolgt und unterdessen eine entkoppelte Entwicklungsarbeit stattfinden kann. Ist während der Entwicklung ein stetiger Modellabgleich oder -austausch mit den unternehmensintern entwickelten Modellen erforderlich, kann darüber hinaus eine Einbindung des externen Lieferanten über spezielle IT-Einwahlknoten bewerkstelligt werden und so ein Entwicklungsprozess mit fortlaufendem Modellabgleich stattfinden. Unternehmensintern ist über das zentrale Datenbanksystem sichergestellt, dass die Zusammenarbeit mehrerer synchron eingesetzter Entwickler koordiniert werden kann. Durch die über die MeMo-Plattform implementierten Algorithmen ist eine Aufteilung von Modellierungselementen in separat versio-

nierbare Einheiten sowie eine Statusdarstellung im Falle der Weiterentwicklung gewährleistet. Ebenso ist in dem Entwicklungswerkzeug durch einen „Model-Info-Block[89]“ sichergestellt, dass Weiterentwicklungen unmittelbar zu erkennen sind. Diese ermöglicht darüber hinaus einen schrittweisen Funktionstest in einer frühen Phase mit MiL, SiL und HiL. Neben der Entwicklung mechatronischer Systeme ist auch in anderen Entwicklungsgruppen die frühzeitige Synchronisierung der Modellbestandteile erforderlich, um im Systemverbund testen und mögliche Fehler früh erkennen zu können.

## 6.4 Applikation am Beispiel in Matlab/Simulink

Basierend auf den Metainformationen (vgl. Abschnitt 5.1.3), die zu jedem Modell hinterlegt sind, wählt der Entwickler die Teilmodelle des mechatronischen Funktionsmoduls aus, die beispielsweise für den modellbasierten Test mittels MiL, SiL oder HiL benötigt werden. Per Drag&Drop und dem Verbinden der Signalflusslinien integriert er diese in einer neuen Modellkonfiguration. Das in Abbildung 6.5 dargestellte mechatronische Funktionsmodul ist aufgebaut aus den Teilelementen *Lenkmechanik (Version 3)*, *EPS_ECU (Version 1)* – dem Regler – und *EPS_ASM_Lenkaktor (Version 2)* – dem Aktor.

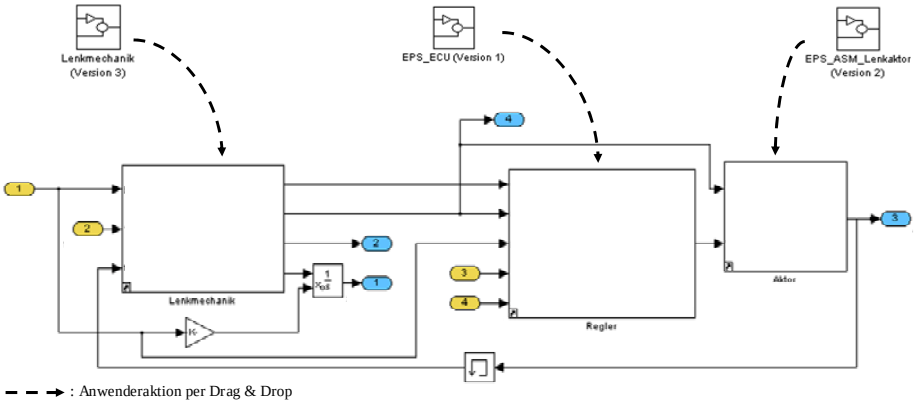


**Abbildung 6.5 - Aufbau eines mechatronischen Funktionsmoduls**

Das daraus entstandene Modell wird mit den enthaltenen Teilelementen versionsgenau als Modellkonfiguration zur Wiederverwendung in der Modellablage abgelegt.
Auf übergeordneter Ebene wird das hierarchisch im Subsystem gegliederte mechatronische Funktionsmodul mit einem Fahrdynamikmodell und weiteren Regelsystemen verknüpft. Abbildung 6.6 stellt das mechatronische Gesamtmodell für den modellbasierten Test mittels SiL dar und ist aufgebaut aus den mechatronischen Teilelementen *ESP (Version 5)*, *EPS (Version 2)* und dem *Fahrzeugstreckenmodell (Version 10)*.

[89] Der Model-Info-Block stellt die Informationen zu Status, Verantwortlicher, Version etc. direkt auf der Oberfläche des Simulink-Modells dar.

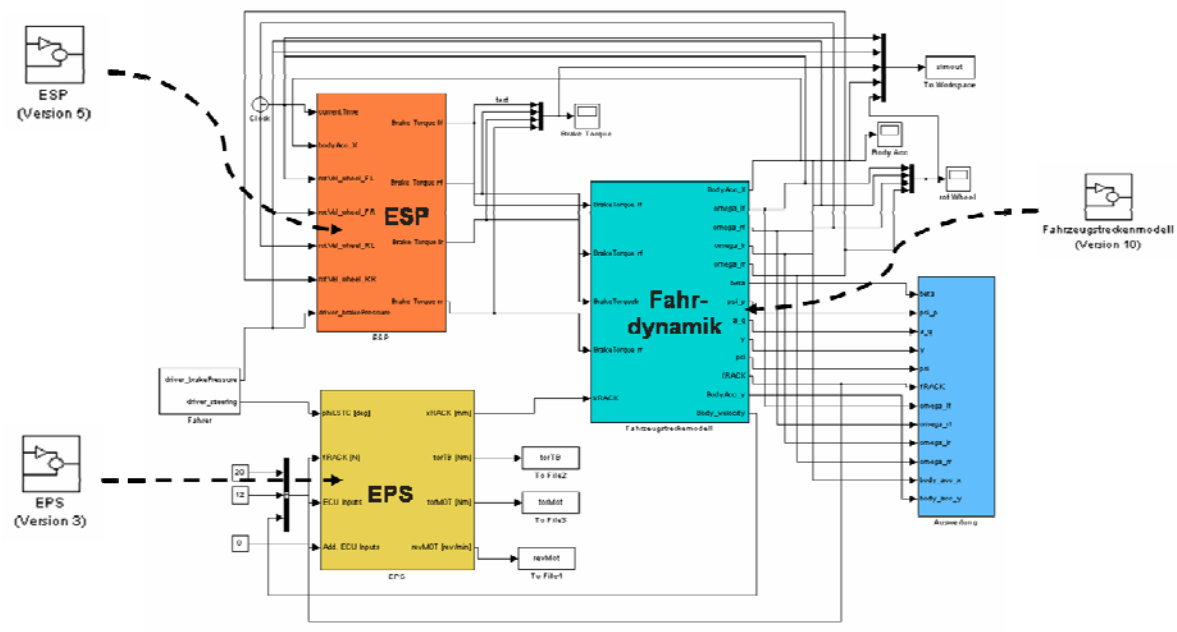


**Abbildung 6.6 - Mechatronisches Fahrzeugmodell mit ESP und EPS zur Absicherung mittels SiL**

Nach dem Verbinden der Signalflusslinien wird das neu entstandene Modell als Modellkonfiguration mit den enthaltenen Teilelementen versionsgenau in dem Datenbanksystem abgelegt und steht für andere Anwender zur Wiederverwendung zur Verfügung. Mit dem in Abbildung 6.6 gezeigten SiL-Simulationsmodell kann die Funktionsabsicherung gemäß der Systemanforderung bzw. -spezifikation mit verschiedenen Fahrmanövern durchgeführt werden, wie dies methodisch im Abschnitt 2.1.3 beschrieben ist. Abbildung 6.7 zeigt beispielhaft die Simulationsergebnisse für ein Testmanöver, um die ABS-Algorithmen, die in der ESP-Funktion unterlagert sind, zu testen.

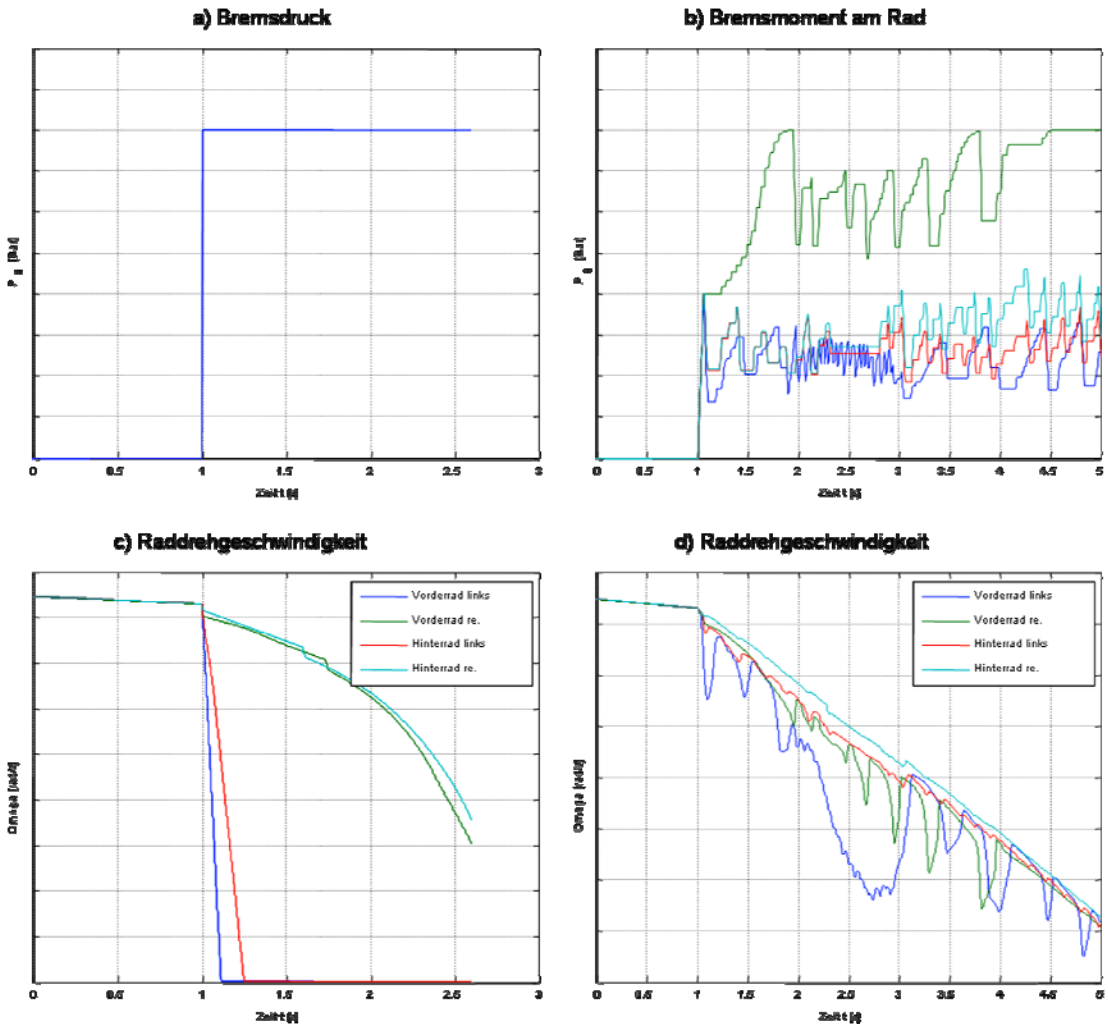


**Abbildung 6.7 - Beispielhaftes Simulationsergebnis zur Absicherung der ABS-Funktion mittels SiL**

Das Diagramm zeigt das Fahrmanöver einer µ-Split-Bremsung, das zum Zeitpunkt 1 Sekunde mit einem Bremsdruck für eine Vollbremsung initiiert wird. Dies führt dazu, dass die Raddrehzahlgeschwindigkeit des passiven Systems (dargestellt in Teilausschnitt c) auf 0 $m \cdot s^{-1}$ absinkt und die beiden Räder mit der geringeren Haftreibungskoeffizienten blockieren. Beim aktiven System, dargestellt in Teilabbildung c, erfolgt eine kontrollierte Verzögerung des Fahrzeugs ohne das Blockieren der Räder. So kann gewährleistet werden, dass das Fahrzeug weiterhin beherrschbar bleibt.

Ein weiteres Beispiel für die Funktionsabsicherung der EPS-Funktion wird mit dem nachstehenden Fahrmanöver durchgeführt. Das System wird mit einem Lenkradwinkel (Lenken und Gegenlenken) mit steigender Lenkradwinkelamplitude und gleichbleibender Frequenz beaufschlagt. Auch hier wird die Simulation als passives (blaue Linien) und aktives System (rote Linien) durchgeführt.

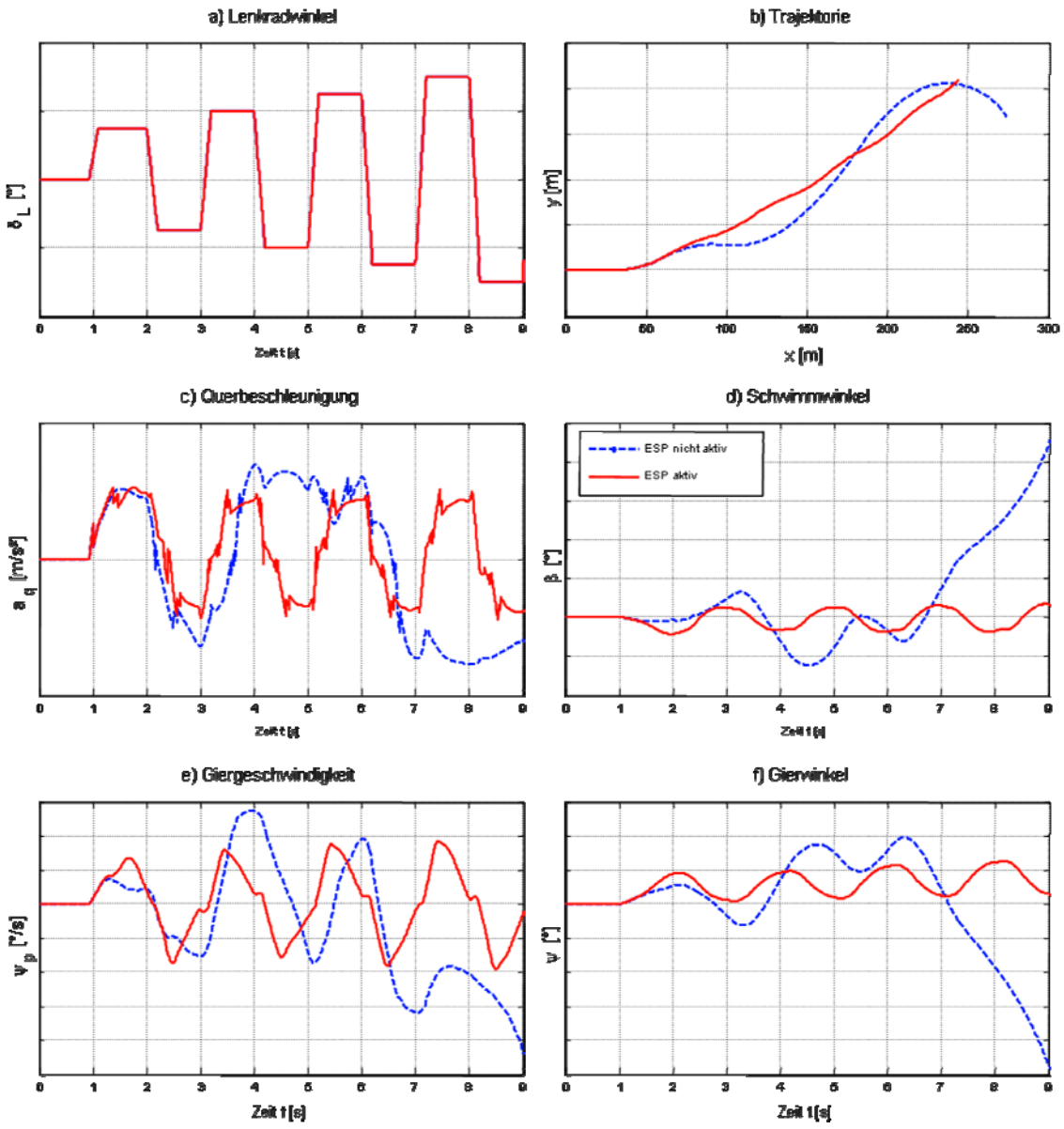


**Abbildung 6.8 - Simulationsergebnis zur Funktionsabsicherung des ESP mittels SiL**

Wie in den Teildiagrammen für die Querbeschleunigung, dem Schwimm- und Gierwinkel, zu erkennen ist, folgt das aktive System den Vorgaben am Lenkradwinkel, während das passive System instabil wird und untersteuernd ausbricht.
Ebenso wie die Modelle und Parametersätze in dem Datenbanksystem archiviert werden, können auch die Fahrmanöver, Testergebnisse und Referenzdaten in dem Datenbanksystem als Referenz oder innerhalb des Modells bzw. Parametersatzes abgelegt und unter Versionskontrolle gestellt werden.

# 7 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde eine offene Integrationsplattform mit dem Namen „MeMo“ zur Unterstützung des mechatronischen Entwurfs von Fahrzeugsystemen vorgestellt. Beginnend mit der Betrachtung generischer und mechatronischer Modelle und einer Analyse vorhandener Werkzeuge für die modellbasierte Entwicklung (vgl. Kapitel 2) wurden die Anforderungen an eine Softwareumgebung definiert. Vor dem Hintergrund der darauf ausgearbeiteten Spezifikationen wurde ein Konzept für die neue Softwareumgebung erarbeitet (vgl. Abschnitt 3.2). Die resultierende Architektur basiert auf einem Kernbaustein - der MeMo-Plattform -, die ein zentrales Datenbanksystem mit den Entwicklungswerkzeugen der Anwender im Umfeld mechatronischer Systeme verbindet. Weiterhin wird durch die MeMo-Plattform die Organisation von CAE-Modellen in einem Datenbanksystem gesteuert. Als Datenbanksystem wurde ein hausinternes System gewählt, in dem durch die MeMo-Plattform eine Steuerung des Versionsmanagements von CAE-Modellen organisiert wird. Im Umfeld rechnergestützter Methoden wird ein Modell stets durch eine Dateigruppe abgebildet. Durch die MeMo-Plattform erfolgt auf der Ebene von Dateigruppen ein zusätzliches Versions-, Konfigurations- und Variantenmanagement. Das Konfigurationsmanagement ermöglicht ein flexibles Arbeiten in Form von Softwarekooperationen über interne Organisationseinheiten hinweg und auch mit externen Lieferanten. Dieses geschieht über ein spezielles Webportal. Dabei ist stets sichergestellt, dass die Entwicklung mechatronischer Systeme in einem integrativen Entwicklungsprozess erfolgen kann und Modelle unter Verwendung eines Variantenmanagements in diversen Projekten eingesetzt werden können.
Als erstes Entwicklungswerkzeug wurde Matlab/Simulink mit der MeMo-Plattform verknüpft. Daraus resultiert eine interaktive Unterstützung direkt in dem vom Anwender gewohnten Entwicklungswerkzeug für den entsprechenden Aufgabenbereich. Das Konzept der MeMo-Plattform sieht eine offene Schnittstelle zur Einbindung weiterer Werkzeuge aus diversen Domänen vor.

Die hier geschaffene Integrationsplattform kann seit Sommer 2007 produktiv eingesetzt werden und somit den Entwickler bei der modellbasierten Auslegung mechatronischer Systeme in entscheidendem Maße unterstützen. Damit kann die MeMo-Plattform dazu beitragen, die heutige Entwicklung mechatronischer Fahrzeugsysteme effizienter zu gestalten.

**Ausblick**

Aus dem Einsatz der Integrationsplattform MeMo mit dem Entwicklungswerkzeug Matlab/Simulink haben sich weitere Wünsche für die Anwendung und Erweiterungen ergeben. Hierzu zählt z. B. die Dokumentation der Modellqualität. Da letztendlich die Qualität des Modells nur nach intensiver Einarbeitung und ausreichender Dokumentation bzw. vom Modellentwickler selbst beurteilt werden kann, ist auf andere Qualitätsmerkmale zurück-

zugreifen. Es können folgende Kriterien herangezogen werden, um eine indirekte Aussage über das Modell zu erhalten:

- Anzahl der Referenzen auf ein Modell
- Anzahl der Nutzer, die das Modell verwenden
- Beurteilung des Modells aufgrund eines allgemeingültigen Richtlinienkataloges (z. B. [VWM06])

Die drei genannten Kriterien sind softwaretechnisch auszuwerten. Eine Beurteilung aufgrund eines Richtlinienkatalogs kann automatisiert mit Hilfe der Matlab-Erweiterung z. B. „Model Advisor“ erfolgen. Weiterhin haben die Schnittstellen bereits bei Projektstart eine zentrale Bedeutung bei der Auswahl von Modellelementen für eine neue Konfiguration. Hier ist ein zusätzlicher Einsatz von z. B. UML[90] , SysML[91] oder AUTOSAR denkbar, um zukünftig die Planung neuer Projekte und den Austausch von Teilmodellen zu unterstützen.

Die interaktive Verknüpfung der MeMo-Plattform an Entwicklungswerkzeuge wie Matlab/Simulink ergibt die Möglichkeit, Testabläufe unter Berücksichtigung diverser Parametersätze durchzuführen. Hier liegt der Fokus auf einer weitergehenden Unterstützung für den Entwickler respektive den Tester, der über MeMo Konfigurationen für den Testablauf aufbauen kann und diese beispielsweise über Nacht unter automatischer Rückspeicherung der Ergebnisdaten durchführen kann. Eine Implementierung kann aufgrund der Architektur der MeMo-Plattform sowohl auf Matlab-Ebene mit einem Zugriff auf die Daten in MeMo als auch mit MeMo unter einem Zugriff auf Matlab als Berechnungsinstrument geschehen.

[90] Unified Modeling Language (UML) ist eine von der Object Management Group entwickelte und standardisierte Beschreibungssprache für die Modellierung von Software. Der Einsatz von UML sowie die Vor- und Nachteile für die Entwicklung mechatronischer Systeme ist in [Ber00], [Hof00], [Sau00] und [Oes05] beschrieben.

[91] Systems Modeling Language (SysML) ist eine auf UML basierende Beschreibungssprache und speziell für die Aufgaben der Systementwicklung (System Engineering) entwickelt worden [Stu06], [Hau07].

# 8 Literaturverzeichnis


[Alb06] Albers, A.; Gschweitl, K.; Schyr, C.; Kunzfeld, S.
Methoden und Werkzeuge zur modellbasierten Validierung von Hybridantrieben
ATZ, 2006, 11, 980-987

[Arn07] Arndt, D.
Eclipse RCP - Entwicklung von Java-Anwendungen für den Desktop (Zugang:11.12.2007)
http://www.contentmanager.de/magazin/artikel_1492_java_eclipse_rcp.html
2007

[Ast98] Åström, K. J.; Elmqvist, H.; Mattsson, S. E.
Evolution of Continous time modeling and simulation
The 12th European Simulation Multiconference, ESM'98, June 16-19, 1998, Manchester, UK

[Bal00] Balzert, H.
Lehrbuch der Software-Technik, Software-Entwicklung
Spektrum Akad. Vlg., 2000, 2. Auflage

[Bay05] Bayer, M.
Konfigurationsmanagement
Automotive Electronics / Sonderheft ATZ/MTZ, 2005, März, 36 – 39

[Bei06] Beine, M.; Eisemann, U.
Vorteile einer toolgestützten Datenhaltung
Elektronik automotive, 2006, 06, 66-70

[Bei07] Beine, M.
Daten-Management mit dem dSpace Data Dictionary
5. dSPACE Anwenderkonferenz 2007

[Bei97] Beitz, W.; Grote, K.; Berger, C.
Taschenbuch für den Maschinenbau - Dubbel
Springer, 1997, 19., völlig neubearb. Aufl.

[Ber00] Bertram, T.; Petersen, J.; Flores, P. T.; Lapp, A.; Walther, M.; Schirmer, J.
Objektorientierte Ordnungssstrukturen mechatronischer Systeme
Konstruktion, 2000, 52, 48-50

[Bos07] Robert Bosch GmbH
Kraftfahrtechnisches Taschenbuch
Vieweg, 2007, 26., überarb. u. erg. A.

[Bre04] Brechter, D.
Implementierung einer ESP-Funktion in ein komplexes Mehrkörpersystem (MKS) für die Gesamtfahrzeugsimulation und Vergleich der Co-Simulation des Gesamtprozesses zur MKS-integrierten Lösung.
Diplomarbeit an der TU Bergakademie Freiberg, 2004

[Bre06] Brechter, D.; Liu-Henke, X.
Eine datenbankbasierte Softwareumgebung zur Unterstützung des mechatronischen Entwurfs
4. Paderborner Workshop: Entwurf mechatronischer Systeme, 2006

[Bre07] Brechter, D.; Liu-Henke, X.
Eine Simulationsumgebung zur Unterstützung der modell-basierten Funktionsabsicherung von Regelsystemen im Kraftfahrzeug
VDI-Berichte Nr. 1971, 2007, 167-177

[Bro05] Bröcker, C.
iX-Studien "Bessere Software!": Konfigurations- und Änderungsmanagement sowie UML
Hannover: Heise, 2005

[Bun04] Bunzel, S.; Judaschke, U.
Process Integration of Model-Based Design and Production-Code Generation in the Multi-User / Multi-Project Development Environment at Continental Teves - Part 1
The Mathworks, Newsletters - Automotive Digest, 2004, 2

[Bus93] VDI-Verlag
Mechatronics in Japan
Buß, M. AND Hashimoto, H., 1993, Reihe 12

[Col04] Ben Collins-Sussman, B.; Pilato, C.
Version control with Subversion : next generation open source version control
O'Reilly, 2004, 1. ed. . - Beijing [u.a.]

[Col05] Collins-Sussman, B.; Fitzpatrick, B. W.; Pilato, M.
Versionskontrolle mit Subversion
O'Reilly Verlag, 2005, 1.Auflage deutsche Ausgabe

[Cre04] Creutzburg, U.; Kalix, E.
Process Integration of Model-Based Design and Production-Code Generation in the Multi-User / Multi-Project Develop-ment Environment at Continental Teves - Part 2
The Mathworks, Newsletters - Automotive Digest, 2004, 2

[Dab04] Dabney, J.
Return on Investment for Independent Verification & Validation
NASA, 2004, Phase 2B Final Report

[Dar02] Darwin, I. F.
Java Kochbuch
O'Reilly, 2002, 2. Auflage

[Dav06] Scholz-Meisner, S.
Handbuch zu DoRIS: Document Retrieval and Information System, Version 2.1
DAVID GmbH, Braunschweig, 2006

[Enc90] Encarnacao, J. L.; Lockemann, P. C.
Engineering Databases - Connecting Islands of Automation through Databases
Springer-Verlag, 1990

[Erl07] Erl, H.-P.; Kirstan, S.
Studie "Kosten-/Nutzenanalyse der modellbasierten Softwareentwicklung im Automobil"
Arthur D Little und TU München. 2007

[Fis83] Fischer, W. E.
Datenbanksystem für CAD-Arbeitsplätze
Fakultät für Maschinenbau der Universität Karlsruhe,
Springer-Verlag, 1983

[Fu06] Fuchs, M.; Schiele, P.; Siwy, R.
Unterstützung der modellbasierten Funktionsentwicklung durch die Modellbibliothek
Modellierung 2006 - 22.03.-24.03.06 - Innsbruck, 2006

[Gam95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.
Design Patterns - Elements of Reusable Object-Oriented Software
Addison-Wesley Professional Computing Series, 1995, 1. Auflage

[Gau00] Gausemeier, J.; Lückel, J.
Entwicklungsumgebung Mechatronik: Methoden und Werkzeuge zur Entwicklung mechatronischer Systeme
Paderborn, 2000

[Gue03] Gühmann, C.
Einsatz der Simulation in der Applikation und im automatisierten Test von Getriebesteuerungen
IIR Tagung Getriebeelektronik, 24./25. Juni 2003

[Gue031] Gühmann, C.
Modellbildung und Simulation für die Funktions- und Softwareentwicklung in der Automobilelektronik
ASIM-Jahrestagung 2003, 2003

[Har04] Harris, R.; Warner, R.
The Definitive Guide to SWT and JFace
Apress, 2004

[Hat04] Hatton, T.
SWT: A Developer's Notebook
O'Reilly, 2004

[Hau07] Hause, M.; Korff, A.
Überblick über SysML für Automobil-Systementwickler
ATZelektronik, 2007, 03, 22-28

[Hei02] Hein, J.
Linux-Systemadministration - Einrichtung, Verwaltung, Netzwerkbetrieb
Addison-Wesley, 2002, 4. Auflage

[Hof00] Hofmann, P. E. H.; Fasolt, J.; Geretschläger, P.; Sakretz, R.; Wohlgemuth, F.
Automotive UML - eine neue objektorientierte Entwicklungstechnik
Elektronik Automotive, 2000, 88-95

[Hol06] Holzenkamp, C.; Sax, E.; Walliser, D.
Testen vernetzter Elektroniksysteme
ATZelektronik, 2006, 04, 26-31

[Hom06] Hommel, M.
Parallelisierte Simulationsprozesse für virtuelles Prototyping in der Automobilindustrie
TU Braunschweig - Institut für Regelungstechnik, 2006

[Hon97] Honekamp, U.
Structuring approach for complex mechatronic systems
Reihe Sonderforschungsbereich / Sonderforschungsbereich 376, Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen; Paderborn, 1997

[IBM06] IBM - Online Handbuch zu IBM Clear Case (Zugang: 08.08.2006)
http://www-306.ibm.com/software/rational

[Ise06] Isermann, R.
Fahrdynamik-Regelung. Modellbildung, Fahrerassistenzsysteme, Mechatronik
Vieweg, 2006, 1. Auflage

[Ise99] Isermann, R.
Mechatronische Systeme
Springer, 1999

[Kal97] Kallenbach, E.; Birli, O.; Saffert, E.; Schäffel, C.
Zur Gestaltung integrierter mechatronischer Produkte
VDI Berichte, 1997, Nr.: 1315, 1-14

[Kit86] Kitaura, K.
Industrial Mechatronics
New East Business Ltd., 1986

[Klo06] Klotzbach, S.; Herfeld, T.
Eine flexibel konfigurierbare Modellumgebung für die Fahrdynamiksimulation
Steuerung und Regelung von Fahrzeugen und Motoren - Autoreg 2006, 1931, 259-270

[Kru00] Krüger, I.
Go To Java 2
Addison Wesley, 2000, 2. Auflage

[Kun04] Kunze, M.
Model based development environment at Siemens VDO
Konferenz-Einzelbericht: IAC 2004, 4. Internat. Automotive Conf., Users of, 2004, 26-35

[Kva06] Kvasnicka, P. et al.
Durchgängige Simulationsumgebung zur Entwicklung und Absicherung von Fahrdynamischen Regelsystemen
VDI-Konferenz: Berechung und Simulation im Fahrzeugbau, 2006, 1967, 387-403

[Lin03] Lindemann, M.; Gühmann, C.
VeLoDyn - Ein Werkzeug zur Triebstrangsimulation von Kraftfahrzeugen
1. Tagung Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik, 2003

[Liu05] Liu-Henke, X.; Duym, S.
Modellgestuetzte Funktionsabsicherung des vernetzten mechatronischen Kraftfahrzeugs
VDI-Verlag, Düsseldorf, 2005, 1073-1090

[Liu05D] Liu-Henke, X.
Mechatronische Entwicklung der aktiven Feder-/Neigetechnik für das Schienenfahrzeug RailCab
VDI-Verlage, 2005

[Mad05] Madsen, B.
Integrating MATLAB with Perl and Java - Examples from Bioinformatics
The Mathworks - MATLAB Digest, 2005

[Mat05] The Mathworks
Documentation to Matlab, Version 7.1
2005

[Mat07] The Mathworks
How do I call back to MATLAB from my Java code that I call using MATLAB's Java interface?
Technical Solutions, 2007, Solution ID: 1-19JDF

[Mca05] McAffer, J.; Lemieux, J.
Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications
Addison Wesley Professional, 2005

[Mer02] Merz, R. M.
Objektorientierte Modellierung thermischen Gebäudeverhaltens
Universität Kaiserslautern, 2002

[Mey07] Meywerk, M.
CAE-Methoden in der Fahrzeugtechnik
Springer, 2007

[Mit04] Mitschke, M.; Wallentowitz, H.
Dynamik der Kraftfahrzeuge
Berlin [u.a.]: Springer, 2004

[MKS06] MKS - Dokumentation zu Source Integrity (Zugang: 30.06.2006)
http://www.mks.com/support/index.jsp?page_id=993

[Mor06] Morgan, G.
Auswirkungen von Autosar auf zukünftige Steuergerätesoftware
ATZelektronik, 2006, 03, 7-13

[Oes05] Oestereich, B.
Analyse und Design mit UML 2: objektorientierte Softwareentwicklung
München, Wien [u.a.]: Oldenbourg, 2005, 7. aktualisierte Auflage

[Pfe96] Pfeifer, T.
Qualitätsmanagement
München [u.a.]: Hanser, 1996

[Pla06] Integrated development process electr(on)ics (internes Papier)
Volkswagen AG - Konzerntagung Produktprozess 2006

[Pre05] Preißel, R.; Stachmann, B.
Subversiv entwickeln?
Java Magazin 05/2005; Software&Support Verlag, Frankfurt, 2005, 113-117

[Ric03] Richert, F.; Rückert, J.; Schloßer, A.
Dymola und Matlab im praktischen Einsatz: Ein Vergleich anhand der Modellierung eines Dieselmotors
ARGESIM Report no.25
"Modellierung und Simulation in Automotive und Prozessautomation" Mismar 8.-9. Mai 2003

[Rot02] Rothfuss, R. et al
Systems engineering in the design of mechatronic systems
International journal of vehicle design, 2002, Vol. 28 Nos.1/2/3 2002, 18-36

[Ruh00] Ruh, W. R.; Maginnis, F. X.; Brown, W. J.
Enterprise Application Integration. A Wiley Tech Brief.
John Wiley & Sons Inc, 2000

[Sau05] Saul, T.; Titze, A.; Bikker, G.; Lawrenz, W.
Grafische Formalisierung von Elektronik-Systemen im Kfz zur Bewertung von Architekturen in frühen Entwicklungsphasen
VDI Berichte Nr. 1907, 2005, 535-562

[Sch05] Schlüter, W.; Kvasnicka, P.; Kämpf, B.
Model Database for Complex Simulink Models
Model-Based Design Conference, 2005

[Schae03] Schäuffele, J.; Zurawka, T.
Automotive Software Engineering - Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen
Friedr. Vieweg & Sohn Verlag, 3., verbesserte und erweiterte Auflage Marz 2006

[Schi04] Schieber, R.; Derichsweiler, F.
Testautomatisierung in der Automobilindustrie
Objektspektrum, 2004, 4, 63-67

[SCM06] Version Control Systems Comparison (Zugang: 09.03.2006)
http://better-scm.berlios.de/comparison/comparison.html

[Stu06] Stuecka, R.
SysML und Fahrzeugelektronik
ATZelektronik, 2006, 03, 48-51

[Teg05] Tegethoff, W.; Kossel, R.; Richter, C.
Modellierung thermischer Systeme in Modelica - Vorlesungsskript TU Braunschweig
2005

[Ull07] Ullenboom, C.
Java ist auch eine Insel
Galileo Computing, 2007, 7., aktualisierte Auflage

[VDI06] VDI - Richtlinie 2206
Entwicklungsmethodik für mechatronische Systeme
Verein Deutscher Ingenieure, 2004

[Vet03] Vettermann, S.
Collaborative Engineering mit durchgängigem Produktdatenmanagement
Proceedings der Online, Düsseldorf, 2003

[VWI05] Vergleich unterschiedlicher Datenbankmanagementsysteme
Volkswagen AG - internes Papier -
2005

[VWM06] Hofmann, Peter-Michael
Modellierungsrichtlinien für MATLAB/Simulink/Stateflow
Volkswagen AG, 2006

[Web02] Webb, P.
MATLAB Programming Patterns Integrating Java Components into MATLAB
The MathWorks - MATLAB News & Notes, 2002

[Wik01] Wikipedia - Tag (Informatik) (Zugang: 25.04.2006)
http://de.wikipedia.org/wiki/Tag_%28Informatik%29

[Wik02] Wikipedia - Versionsverwaltung (Zugang: 04.10.05)
http://de.wikipedia.org/wiki/Versionsverwaltung

[Wik03] Wikipedia - Autosar (Zugang: 22.12.2007)
http://de.wikipedia.org/wiki/Autosar

[Wik04] Wikipedia - Enterprise Java Beans (Online-Zugriff: 13.02.2008)
http://de.wikipedia.org/wiki/Enterprise_JavaBeans

# 9 Anhang

## 9.1 Tabellarischer Vergleich zu Datenbanken im Stand der Technik

| Kriterium | VeLoDyn [Lin03], [Gu031] | Modellbibliothek [Fu06] | Flexible konfigurierbare Modellumgebung [Klo06] |
|---|---|---|---|
| **Anwendungsgebiet (Fokus der gespeicherten Elemente)** | Triebstrangsimulation | Austausch von Modellen mit Lieferanten | Fahrdynamiksimuation |
| *Modellablage* | | | |
| **Zentrale Ablage der Daten (DB)** | Keine zentrale Ablage | Keine zentrale Ablage | Keine zentrale Ablage |
| **Verwendetes Datenbanksystem** | kein Datenbanksystem | kein Datenbanksystem | kein Datenbanksystem |
| **Struktur der Datenbank** | nicht vorhanden | kein Datenbanksystem | kein Datenbanksystem |
| **Lokale Organisation der Dateien** | Dateien werden in einem Träger-Block "Velodyn-Block" organisiert. | keine Aussage | Komponentenbasierte Ablage in Verzeichnissen und Dateien |
| *Versionsmanagement* | | | |
| **Versionkontrolle für einzelne Dateien** | nein | ja | nein |
| **Versionkontrolle für Dateigruppen** | gebündelt durch den VeLoDyn-Block, der selbst keine Versionkontrolle hat | keine Aussage | nein |
| *Konfigurationsmanagement* | | | |
| **Konfigurationsmanagement** | nein | Wird als Aufgabe der Modellbibliothek genannt, die Funktion ist jedoch nicht beschrieben. | nein |
| **Konfigurationsmanagement für Dateigruppen** | nein | keine Aussage | nein |
| **Single-Source-Prinzip** | Hinfällig da keine DB | keine Aussage | Hinfällig da keine DB |
| *Variantenmanagement* | | | |
| **Branching** | keine Aussage | Wird als Aufgabe der Modellbibliothek genannt, die Funktion ist jedoch nicht beschrieben. | Varianten für Modelle werden bereitgestellt. |
| **Aufbau von Referenzen** | keine Aussage | keine Aussage | keine Aussage |
| **Abbildung von Datenvarianten** | Gebündelt durch den VeLoDyn-Block. | Wird als Aufgabe der Modellbibliothek genannt, die Funktion ist jedoch nicht beschrieben. | Nur auf Dateiebene mit unterschiedlichen Parameterdateien. |
| *Metadaten* | | | |
| **Defintion und Eigenschaften von Metadaten** | nicht vorhanden | Reine Dokumentation durch separate Datei möglich. | nicht vorhanden |
| **Metadaten zu einer einzelnen Datei** | Reine Dokumentation durch separate Datei möglich. | keine Aussage | nicht vorhanden |
| **Metadaten auf Dateigruppe** | Reine Dokumentation durch separate Datei möglich. | keine Aussage | nicht vorhanden |
| *Integrativer Entwicklungsprozess ( Softwarekooperation)* | | | |
| **Mehrbenutzerbetrieb** | nein | nein, keine Aussage | nein |
| **Arbeiten in Softwarekoorperationen** | nein | nein | nein |
| *IT-Rahmen* | | | |
| **Berechtigungssystem auf die Elemente** | nein | keine Aussage | nein |
| **Einbindung in die Entwicklungswerkzeuge des Anwenders** | Integration in Matlab/Simulink | Modellothek ist nur zum Austausch der Modelle konzipiert | Als Simulink Bibliothek voll vorhanden |
| **Einbindung in das IT-Prozessumfeld** | keine Aussage | keine Aussage | keine Aussage |

**Tabelle 9.1 - Übersicht zu Softwareumgebungen (Teil 1)**

| Kriterium | BMW<br>[Sch05]; Anwendung mit ISAR: [Kva06] |
|---|---|
| **Anwendungsgebiet (Fokus der gespeicherten Elemente)** | Derzeitige Anwendung für die Entwicklung im Anstriebsstrang und Fahrwerk |
| *Modellablage* | |
| **Zentrale Ablage der Daten (DB)** | vorhanden |
| **Verwendetes Datenbanksystem** | keine Aussage |
| **Struktur der Datenbank** | Struktur:<br>- generelle Werkzeuge<br>- Parameter<br>- modellspezischer Zubehör<br>- Modelle<br>Aus der o.g. Struktur werden Teilelemente in Modulen/Paketen(XML-Datei) zusammengeführt, die separat abgelegt werden. |
| **Lokale Organisation der Dateien** | Die in einem Datenbank als Package zusammengefassten Dateien sind in ein frei definierbares Verzeichnis herunterzuladen. |
| *Versionsmanagement* | |
| **Versionkontrolle für einzelne Dateien** | ja |
| **Versionkontrolle für Dateigruppen** | Dateien werden in Packages als Gruppen zusammengefasst. Ein Package stellt eine XML-Datei als Konfigurationsliste dar. In dieser werden die Versionen der einzelnen Teildateien festgehalten und Metadaten dokumentiert. |
| *Konfigurationsmanagement* | |
| **Konfigurationsmanagement** | Konfigurationsmanagement wird durch XML-Dateien als Konfigurationslisten bewerkstelligt. Diese dienen als Datenschnitttstelle für den Datentransport und die Versionierung. |
| **Konfigurationsmanagement für Dateigruppen** | Packages lassen sich hierarchisch gliedern. |
| **Single-Source-Prinzip** | Ja, durch Referenzierung der Elemente in der Package-Datei. Datenbankmechanismen für Einchecken, Lock etc. sind vorhanden. |
| *Variantenmanagement* | |
| **Branching** | Ja, möglich mit Merge nach dem Vergleich. |
| **Aufbau von Referenzen** | Packages können hierarchisch ineinander gegliedert werden. Referenzen zwischen Modell und Parameterdateien bestehen nur über die flache Liste im Package (XML-Datei).<br>Eine Suche nach verfügbaren Parametersätzen auf einem Modell oder umgekehrt ist nicht möglich. |
| **Abbildung von Datenvarianten** | Datenvarianten können nicht auf Modelle abgebildet werden. Eine Datenvariante bedeutet ein neues Package. |
| *Metadaten* | |
| **Defintion und Eigenschaften von Metadaten** | Metadaten wie:<br>- Status<br>- Release<br>- Beschreibung Modellursprung<br>- Richtlinien<br>- etc.<br>sind vorhanden und werden auf Dateigruppen (Packages) angewendet |
| **Metadaten zu einer einzelnen Datei** | Keine Metadaten vorhanden. |
| **Metadaten auf Dateigruppe** | Metadaten auf Packages(Dateigruppe) sind vorhanden. |
| *Integrativer Entwicklungsprozess ( Softwarekooperation)* | |
| **Mehrbenutzerbetrieb** | Mechanismen in der DB vorhanden. Keine Aussage über die Nutzungsszenarien im Umgang mit den in den Packages referenzierten Elementen. |
| **Arbeiten in Softwarekoorperationen** | Keine Aussage über automatische Updates, Aktionen bei Statusänderung etc. |
| *IT-Rahmen* | |
| **Berechtigungssystem auf die Elemente** | Berechtigungssystem unterscheidet zwischen Anwender und Entwickler. |
| **Einbindung in die Entwicklungswerkzeuge des Anwenders** | Erfolgt über eine separate Toolbox in Matlab "Build-Toolbox", diese unterstützt den Anwender beim Arbeiten in Matlab/Simulink und managet versionen/Elemente/Packages. |
| **Einbindung in das IT-Prozessumfeld** | keine Aussage |

**Tabelle 9.2 -Übersicht zu Softwareumgebungen (Teil 2)**

| Kriterium | Continental Teves [Cre04], [Bun04] |
|---|---|
| **Anwendungsgebiet (Fokus der gespeicherten Elemente)** | Reine Steuergerätecode-Entwicklung bei Continental Teves (z.B.: Bremssysteme etc.) |
| *Modellablage* | |
| **Zentrale Ablage der Daten (DB)** | Als "Project Codebasis" und "Project Configuration" als |
| **Verwendetes Datenbanksystem** | keine Aussage |
| **Struktur der Datenbank** | Aufteilung in "Project Codebasis" für den generischen Funktionsanteil und "Project Configuration" für den Auftragsspezifischen Anteil. Eine Struktur der Datenbasis wurde nicht erläutert. |
| **Lokale Organisation der Dateien** | Durch das die Konfigruations mittles *.h-Datei werden die Elemente lokal konfiguriert. Das Herunterladen aus der Datenbank erfolgt in eine lokale "Sandbox" beim Anwender |
| *Versionsmanagement* | |
| **Versionkontrolle für einzelne Dateien** | ja |
| **Versionkontrolle für Dateigruppen** | keine Aussage |
| *Konfigurationsmanagement* | |
| **Konfigurationsmanagement** | Das Konfigurationsmanagement für die generischen Regelalgorithmen erfolgt durch eine GUI, die nicht näher beschrieben ist. Beim Herunterladen wird automatisch eine Header-Dateien(*.h) generierte, in denen die Teilelemente mit #define spezifiziert/konfiguriert werden. |
| **Konfigurationsmanagement für Dateigruppen** | keine Aussage |
| **Single-Source-Prinzip** | Über das Konfigurationsmanagement erfolgt die Aufteilung in "Project Codebases" und "Project Configuration". Es wird keinen separate Versionierung von Modell und Parametern beschrieben. Die Multi-Project Fähigkeiten sollen künftig weiter ausgebaut werden |
| *Variantenmanagement* | |
| **Branching** | keine Aussage |
| **Aufbau von Referenzen** | Eine Abbildung von Datenreferenzen auf Modelle wird nicht genannt. |
| **Abbildung von Datenvarianten** | Konzept vorhanden, Realisierung wird in Zukunft durchgeführt. "Strap-begin" und strap-end"-block sollen in Simulink das Variantenmanagement durch Aktivierung von Parameterblöcken übernehmen |
| *Metadaten* | |
| **Defintion und Eigenschaften von Metadaten** | keine Aussage |
| **Metadaten zu einer einzelnen Datei** | keine Aussage |
| **Metadaten auf Dateigruppe** | keine Aussage |
| *Integrativer Entwicklungsprozess ( Softwarekooperation)* | |
| **Mehrbenutzerbetrieb** | Mechanismen wie das Auschecken oder das Sperren von Dateien sind implementiert durch Source-Code-Verwaltung (für einzelne Dateien). |
| **Arbeiten in Softwarekoorperationen** | Datenbanksystem zugrundliegende Konfigurationsmanagement sind keine automatischen Updates zu erwarten. |
| *IT-Rahmen* | |
| **Berechtigungssystem auf die Elemente** | keine Aussage zu Berechtigungen |
| **Einbindung in die Entwicklungswerkzeuge des Anwenders** | keine Aussage |
| **Einbindung in das IT-Prozessumfeld** | Verknüpfung der Modelle mit dem Anforderungsmanagement in Doors beschreiben. |

**Tabelle 9.3 - Übersicht zu Softwareumgebungen (Teil 3)**

| Kriterium | dSpace DataDictionary<br>[Bei06], [Bei07] |
|---|---|
| **Anwendungsgebiet (Fokus der gespeicherten Elemente)** | Unterstützung für die Steuergerätecode-Entwicklung<br>Einsatz im Rahmen der Seriencodegenerierung mit TargetLink.<br>Dient als Container, um im Rahmen der Funktionsentwicklung die Daten aufzunehmen. |
| *Modellablage* | |
| **Zentrale Ablage der Daten (DB)** | Keine zentrale Datenbank vorhanden. Lokale Kopien der Objekte |
| **Verwendetes Datenbanksystem** | kein Datenbanksystem |
| **Struktur der Datenbank** | Im DatenDictionary werden Modell-Parameter und Softwarebezogene Daten, wie sie tyischerweise im Rahmen der Funktionsentwicklung und Code-Generieerung für Simulink-Modelle auftreten.<br>Weiterhin werden Betriebssystemobjekte wie Tasks, Events, Ressourcen und Schutzmechanismen<br>Standardstruktur für Elemente. Keine Aussage über die Strukturierung der Datenbank selbst. |
| **Lokale Organisation der Dateien** | Organisation wird durch DataDictionary in einer statisch strukturierten "Sandbox" vorgenommen. |
| *Versionsmanagement* | |
| **Versionkontrolle für einzelne Dateien** | nein |
| **Versionkontrolle für Dateigruppen** | nein |
| *Konfigurationsmanagement* | |
| **Konfigurationsmanagement** | Keine Konfigurationsmanagamanagementfunktionalität<br>- keine Versionskontrollfunktionalität<br>- File-basierte Versionskontrolle über reales Konfigurationmanagement System möglich. |
| **Konfigurationsmanagement für Dateigruppen** | nicht möglich |
| **Single-Source-Prinzip** | Ausschließlich für die Parameter und Einstellungen der Modelle vorhanden |
| *Variantenmanagement* | |
| **Branching** | keine Aussage |
| **Aufbau von Referenzen** | Datenvarianten werden über das DataDictionary als Referenz zu dem Modell abgelegt. |
| **Abbildung von Datenvarianten** | Datenvariantenmanagement durch das DataDictionary selbst basierend auf das Modell. |
| *Metadaten* | |
| **Defintion und Eigenschaften von Metadaten** | Jeden im DataDictionary-Objekt ist eine genaue Menge von festen aber obligatorischen Attributen zugeordnet. |
| **Metadaten zu einer einzelnen Datei** | Metadaten werden nicht auf einzelen Dateien vergeben, sondern sind für die einzelenen Parameter verfügbar und ermöglichen somit eine Dokumentation dieser. |
| **Metadaten auf Dateigruppe** | Keine Dateigruppen vorhanden. |
| *Integrativer Entwicklungsprozess ( Softwarekooperation)* | |
| **Mehrbenutzerbetrieb** | Kein Mehrbenutzerbetrieb im konventionellen Sinn. |
| **Arbeiten in Softwarekoorperationen** | - Keine Nutzer Accounts und Nutzer Gruppen<br>- Aber: Administrator kann schreibgeschützte Objekte definieren |
| *IT-Rahmen* | |
| **Berechtigungssystem auf die Elemente** | Keine Nutzeraccounts und keine Nutzergruppen.<br>Administrator kann schreibgeschützte Objekte definieren<br>Modellparameter können als Standard zentral durch den Admin definiert werden. Zugriffsberechtigungen für Modifizierung etc sind vorhanden. |
| **Einbindung in die Entwicklungswerkzeuge des Anwenders** | Volle Einbindung des Werkzeugs in Matlab/Simulink. |
| **Einbindung in das IT-Prozessumfeld** | keine Aussage |

**Tabelle 9.4 -Übersicht zu Softwareumgebungen (Teil 4)**

| Kriterium | Siemens PowerTrain **[Kun04]** |
|---|---|
| **Anwendungsgebiet (Fokus der gespeicherten Elemente)** | Entwicklungsumgebung in der Antriebsstrangentwicklung |
| *Modellablage* | |
| **Zentrale Ablage der Daten (DB)** | Propriestäres Konfigurationsmanagementwerkzeug |
| **Verwendetes Datenbanksystem** | keine Aussage |
| **Struktur der Datenbank** | keine Aussage |
| **Lokale Organisation der Dateien** | Auf der Festplatte jedes einzelnen Entwicklers wird eine statische definiterte Verzeichnisstruktur erzeugt in die die Dateien heruntergeladen werden (Bezeichnung (Matlab Development Space - MDS)) |
| *Versionsmanagement* | |
| **Versionkontrolle für einzelne Dateien** | ja |
| **Versionkontrolle für Dateigruppen** | Eine Versionierung von Dateigruppen wird nicht genannt. Dateigruppen werden über lokale XML-Dateien erzeugt. |
| *Konfigurationsmanagement* | |
| **Konfigurationsmanagement** | Die beim Modellaufbau verwendeten Dateien bzw. referenzierten Dateien werden in einer XML-Datei festgehalten. Die Struktur der XML-Datei bildet die Konfiguration eines Modells ab und enthält die Dateinamen der referenzierten Elemente, den Speicherort in dem lokalen MDS sowie die Versionen der Dateien.<br><br>*Propriestäres XML-Format für das Konfigurationsmanagementwerkzeug*<br>Konfigurationsmanagement findet in einer vom CM-Werkzeug unabhängigen XML-Datei statt. In dieser wird festgehalten:<br>- Dateien mit dem lokalen Pfad<br>- Versionsnummer der Datei<br>Datei wird in das Konfigruationsmanagementwerkzeug importiert. |
| **Konfigurationsmanagement für Dateigruppen** | Die beschrieben Vorgehensweise schließ eine hierarchische Gliederung von XML-Dateien aus. |
| **Single-Source-Prinzip** | unklar aufgrund der fehlenden Beschreibung, wie mit dem CM bzw. Versionwerkzeuge umgegangen wird. Beschrieben wurden nur Aktionen auf den lokalen Dateien, ohne ein Zugriffs bzw. MultiUser-Konzept zu beschreiben<br>nicht für Dateigruppen; nur die einzelne Dateien |
| *Variantenmanagement* | |
| **Branching** | keine Aussage |
| **Aufbau von Referenzen** | keine Aussage |
| **Abbildung von Datenvarianten** | keine Aussage |
| *Metadaten* | |
| **Defintion und Eigenschaften von Metadaten** | Der Artikel beschreibt die Dokumentation eines Modells/Elements auf Dateiebene, die durch Matlab-Tools erzeugt wird. |
| **Metadaten zu einer einzelnen Datei** | keine Aussage |
| **Metadaten auf Dateigruppe** | Separate Datei für die Beschreibung einer Dateigruppe (Modul). |
| *Integrativer Entwicklungsprozess ( Softwarekooperation)* | |
| **Mehrbenutzerbetrieb** | Versionwerkezuge umgegangen wird. Beschrieben wurden nur |
| **Arbeiten in Softwarekoorperationen** | Eine Kommunikation mit dem Konfigurationsmanagement/Versionsmanagement wird immer vom Anwender angestoßen. |
| *IT-Rahmen* | |
| **Berechtigungssystem auf die Elemente** | keine Aussage zu Berechtigungen |
| **Einbindung in die Entwicklungswerkzeuge des Anwenders** | Mittels des "Matlab Integration Space" werden Pfade in Matlab bereitgestellt und Bibliotheken in Simulink aufgebaut. |
| **Einbindung in das IT-Prozessumfeld** | Modelle sind mit den Anforderungen des Auftragsgebers und Testvektoren (in Zukunft) verknüpft. |

**Tabelle 9.5 -Übersicht zu Softwareumgebungen (Teil 5)**

## 9.2 Referenzen zwischen Modell- und Parametersatzkonfiguration

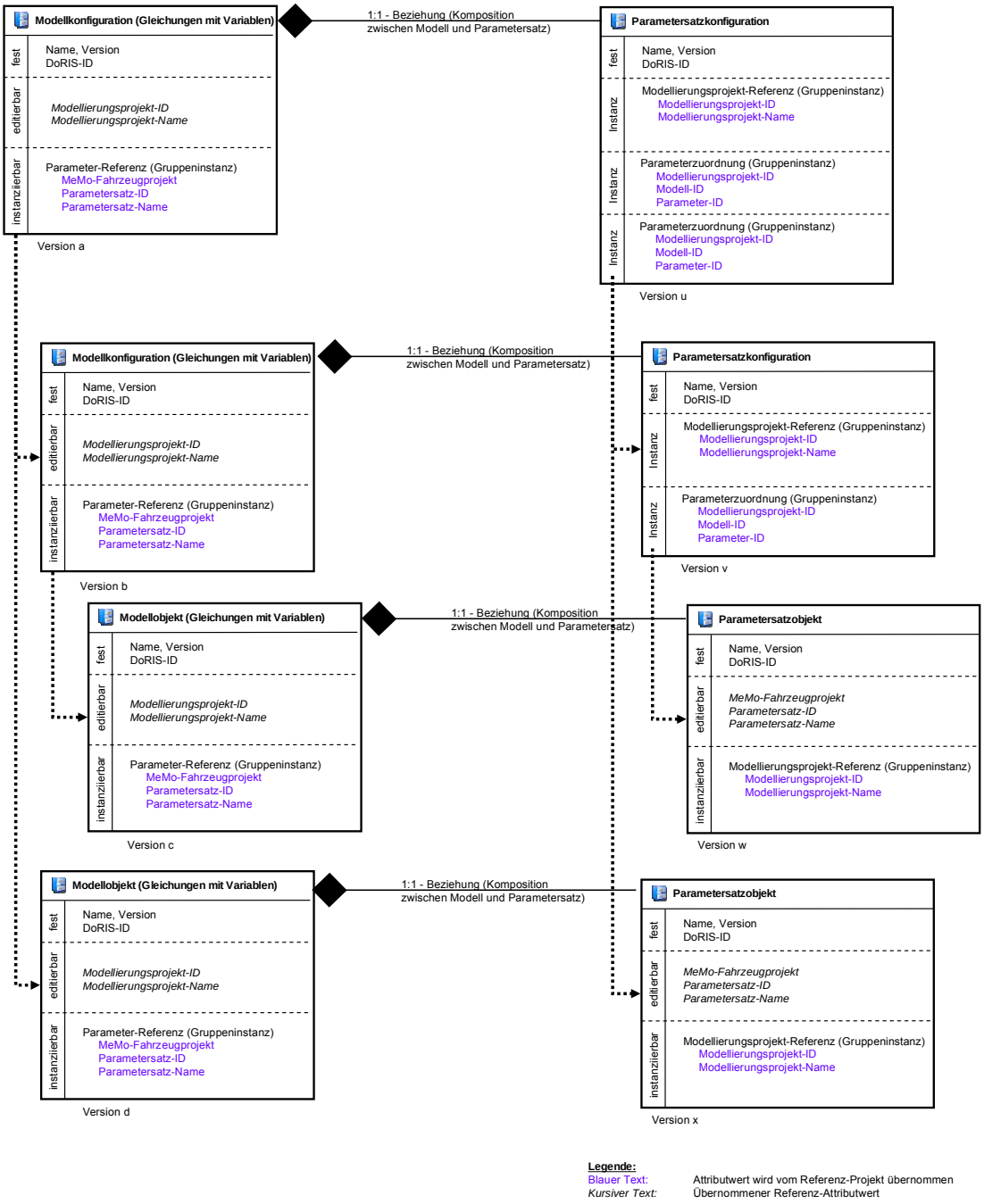


**Abbildung 9.1 - Abbildung der Referenzen zwischen Parametersatzkonfiguration und Modellkonfiguration**